TALQ Specification

Version 2.4.0

December 15th, 2021

About TALQ

The TALQ Consortium has developed a global standard protocol to enable Central Management Software to configure, control, command and monitor multiple Outdoor Device Networks from various suppliers through an easy-to-integrate RESTful/JSON protocol. TALQ is open to industry members to join and participate in the evolution of the TALQ Protocol.
TALQ also provides a Partner Program for cities, municipalities, utilities and consultants to contribute to the future of the Smart City. To learn more about us, our members and partners, please visit www.talq-consortium.org

Copyright

About this document

This document helps TALQ members understand the technical specifications of the TALQ Smart City Protocol. The complete technical description of the TALQ Smart City Protocol is made of:

- This document,
- The TALQ Open API file that describes the RESTful API to be implemented in a Central Management Software to make it TALQ-compatible,
- The TALQ Open API file that describes the RESTful API to be implemented in the Gateway of an Outdoor Device Network to make it TALQ-compatible,
- The Open API file that describes the TALQ data model in detail,
- The list of TALQ functions and Attributes available on the TALQ web site.

The above material is available in the members' area of the TALQ Consortium website.
In case of any conflicting information between the above files, the Open API files supersede this document.
For any further explanation of the contents of this document, or in case of any perceived inconsistency or ambiguity of interpretation, please contact the TALQ Consortium:

- eMail: info@talq-consortium.org
- Web site: talq-consortium.org

Disclaimer

The information contained herein is believed to be accurate as of the date of publication, however the copyright holder will not be liable for any damages, including indirect or consequential from use of or reliance on the accuracy of this document.

If you have any questions regarding the content of this document, please contact the TALQ consortium at info@talq-consortium.org

Terms of use

The document is created and maintained by the TALQ Consortium. Use of this document is governed by the TALQ Consortium Agreement, under which this document has been obtained.

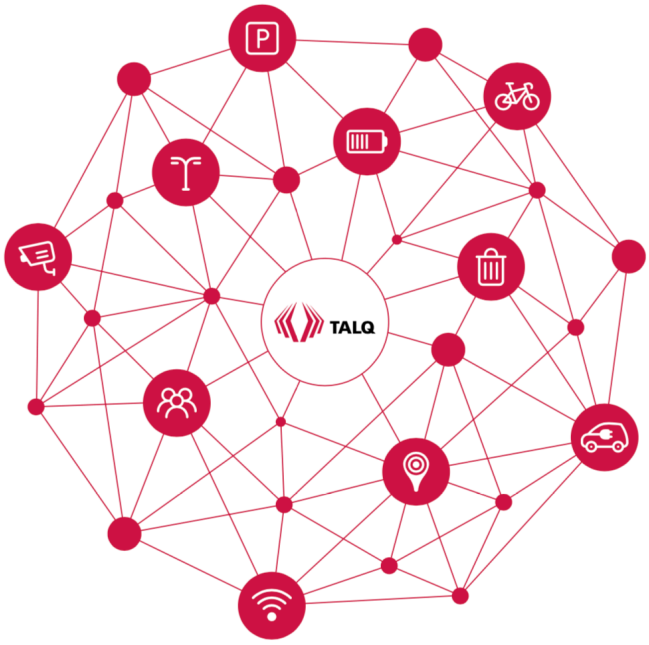
Table of Contents

1 ABOUT TALQ, THE SMART CITY PROTOCOL

1.1 The Challenge

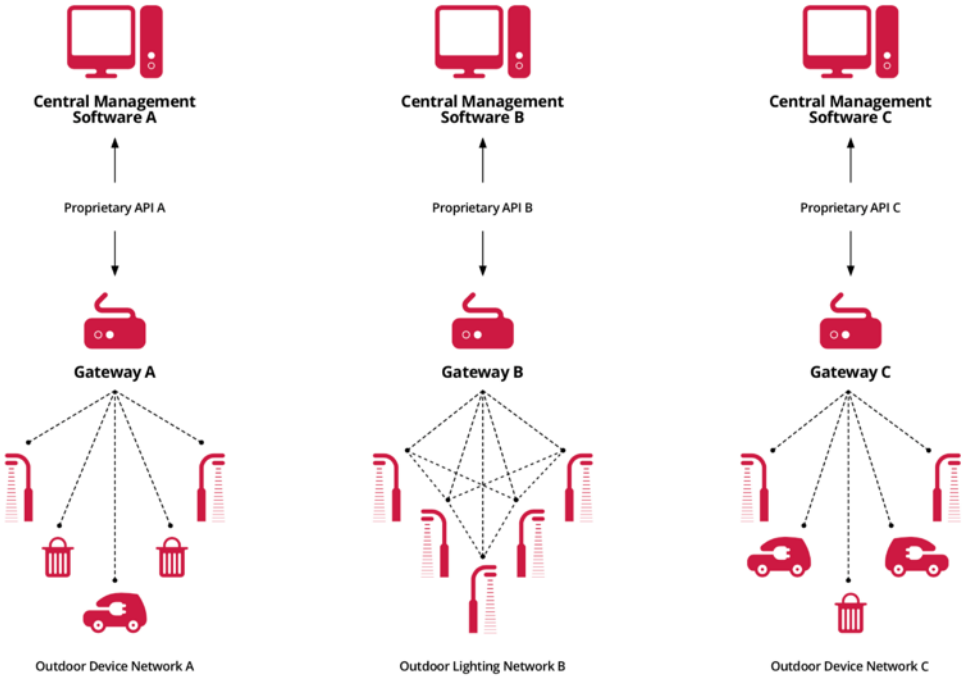
Every day a new Smart City project is announced: streetlights are being connected to save energy and increase the quality of lighting services in the streets, waste containers are being monitored to reduce truck traffic whilst helping cities to get cleaner, free parking spaces are detected and advertised to drivers both to reduce pollution and to allow variable pricing depending on their availability, and more.

Most of the available solutions, however, are proprietary, locking cities into single vendor solutions. Thanks to the TALQ Smart City Protocol, cities can now choose and adopt control solutions from multiple vendors and control them all through a single Central Management Software.



The TALQ Smart City Protocol is an application interface to exchange data, commands and programs between one or more Central Management Software (CMS) and Outdoor Device Networks (ODNs) from different vendors to enable configuration, control, command and monitoring of connected devices in the city.

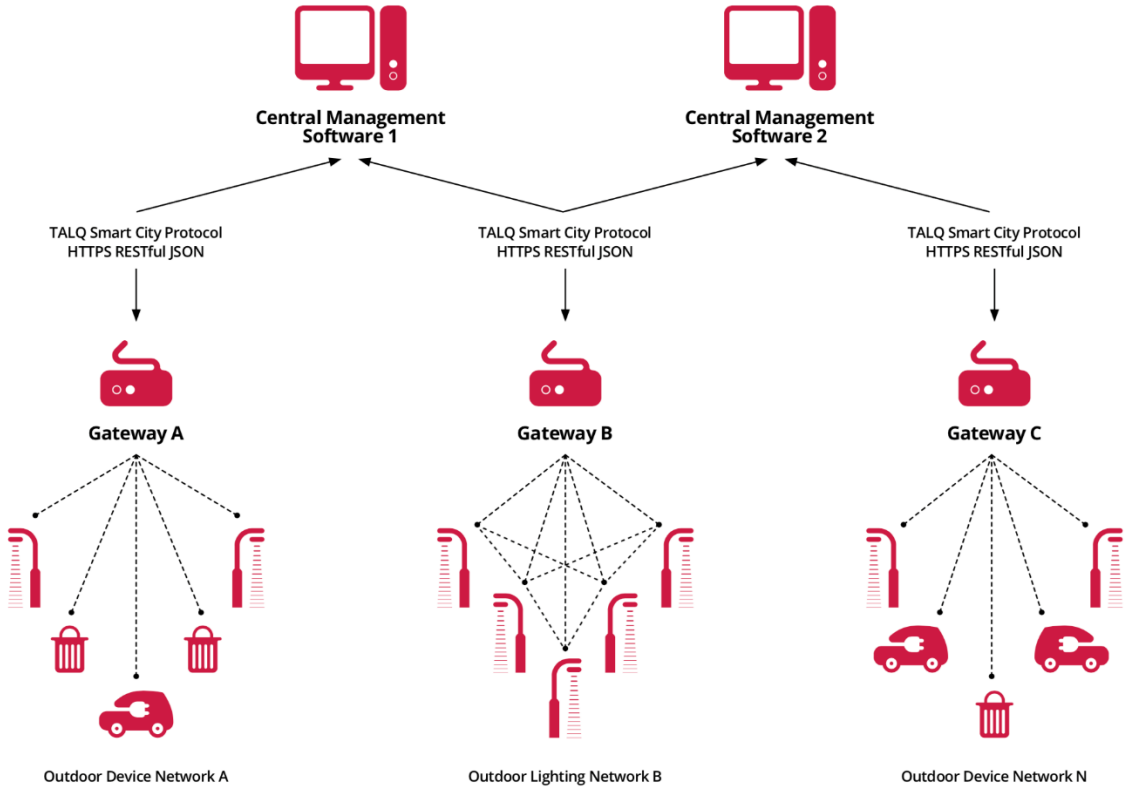
Today's challenge is the complexity associated with multiple proprietary systems and interfaces. The lack of standards makes controlling different connected devices and smooth deployment of various systems for different applications across a region or city very difficult.



Centrally Managed Smart City System without TALQ

1.2 TALQ, the Smart City Protocol

The TALQ Consortium develops and manages the TALQ Smart City Protocol. The TALQ Smart City Protocol defines the message types, data format, parameters and behavior of CMS and Gateways, to configure, control, command and monitor different types of connected devices on various ODNs. As illustrated in the drawing below, TALQ does not constrain the protocols and network implementation of the ODN itself but provides a common interface between an interface to an ODN, called the Gateway, and a CMS.



Centrally Managed Smart City System with TALQ

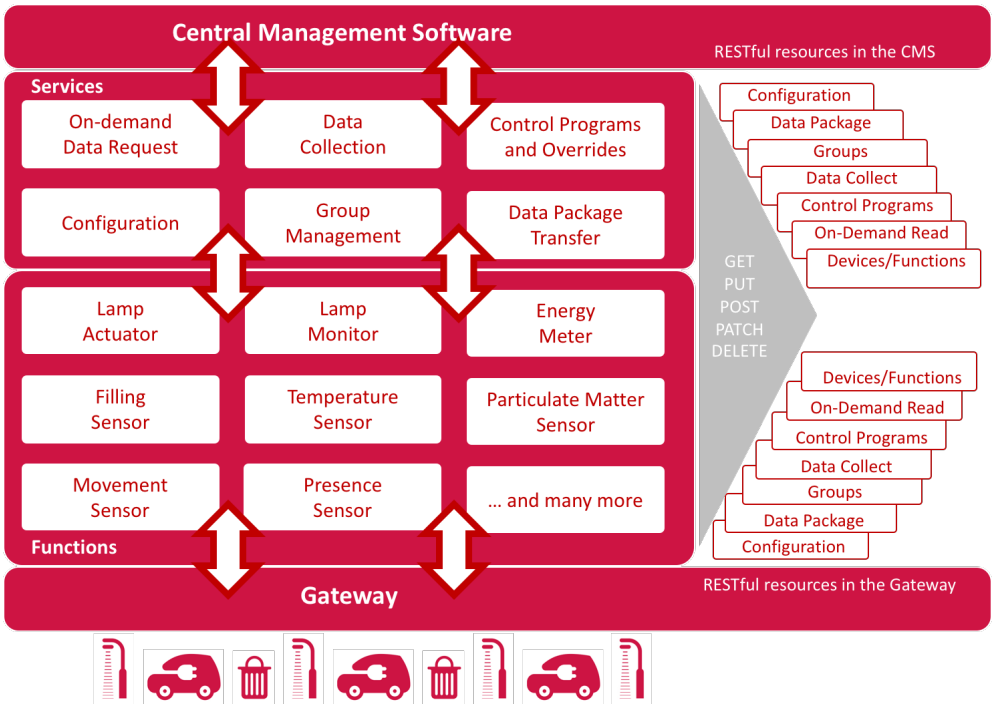
The TALQ Smart City Protocol is designed to offer the following main benefits:

- **TALQ offers a flexible data model** that is applicable to a wide range of sectors and use cases, such as outdoor lighting control, waste collection, parking space detection, environmental data collect, energy management and more. With TALQ, vendors are free to describe their devices thanks to TALQ functions that include a set of agreed configuration, operational and metering attributes and events, which can be configured, controlled, commanded and monitored using TALQ services.
- **TALQ covers a broad set of services**, not only data collection but also configuration services, dynamic control programs and manual overrides, on-demand read service, group management and firmware updates.
- **TALQ is based on well-known standards to be easy to implement**. The standard RESTful approach adopted by TALQ makes it easy to integrate both in existing CMS and Gateways. To enable configuration, control, command and monitoring from a CMS, the TALQ Smart City Protocol provides secured HTTP REST GET, PUT, POST, PATCH and DELETE requests and associated JSON data payloads to describe the devices, their functions and attributes.
- **TALQ provides a comprehensive certification program** and associated test tools which are valuable both for vendors during their implementation process and for end-customers to make sure products are fully interoperable.

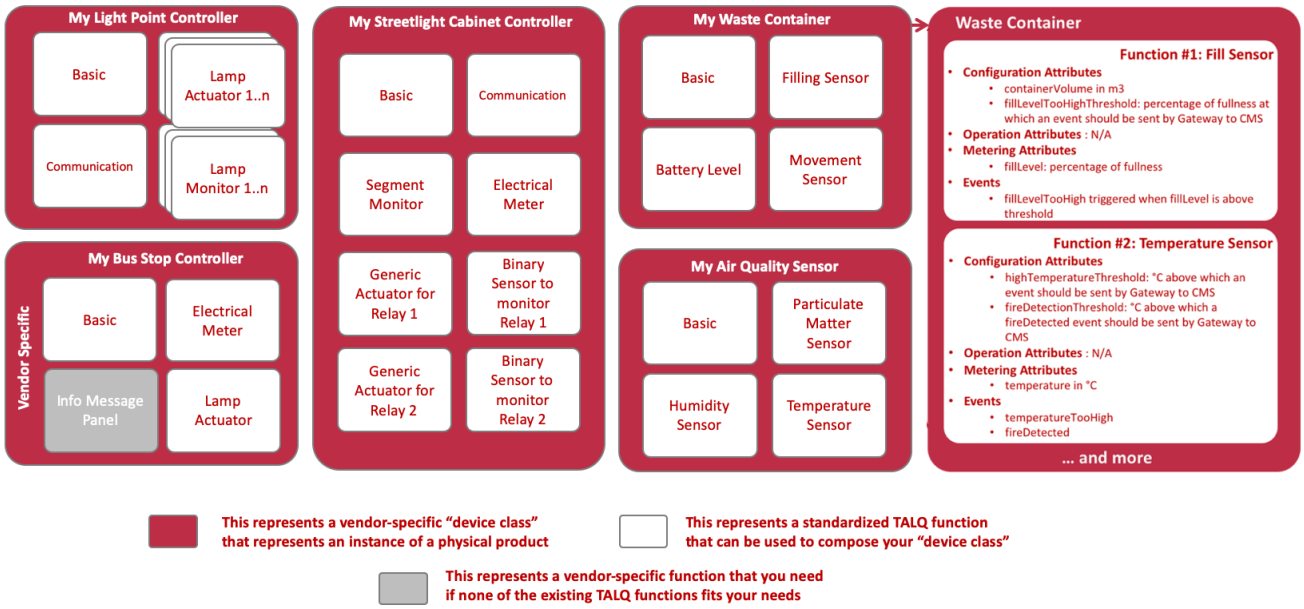
1.3 TALQ, a flexible Smart City Device Data Model

To support many Smart City use cases, including, but not limited to, outdoor lighting control, waste collection, parking space detection, environmental data collect and energy management, the TALQ model:

- provides a whole range of services, not limited to data collect but also including remote control, device configuration, schedulers and calendars, firmware update and more, and
- enables vendors to describe any type of device using functions, attributes and events, while being obvious to understand for Central Management Software.



A TALQ device is made of a set of functions which represents the capabilities of a physical device. Vendors are free to model their physical device using the existing TALQ functions and their attributes and events. They can also use the TALQ generic functions or design their own function, attributes and events by extending existing ones. The diagram below shows typical devices that can be built up as a set of functions. Each function has four types of attributes: configuration, operational (i.e. commands), metering and status (corresponding to events).



Experts in new smart city vertical segments are welcome to work with experts from the TALQ consortium to understand how their use cases and devices can map onto the TALQ device data model. It is likely that this will simply require new functions with associated attributes and events. The set of services available in TALQ is comprehensive and designed in a flexible way to support many different use cases.

Function Name : _____

Description : _____

_____

Configuration Attributes

Attribute name	Type	Unit	Description
	Boolean		
	Float		

Operation Attributes

Attribute name	Type	Unit	Description
	Boolean		
	Float		

Metering Attributes

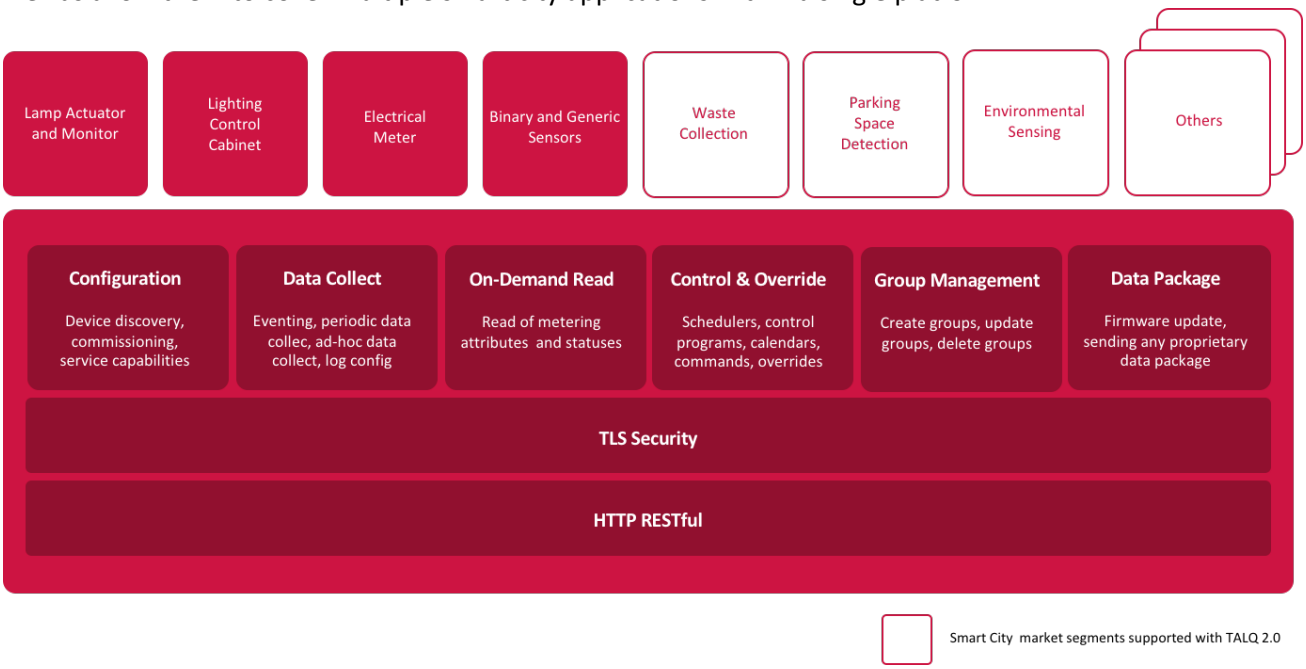
Attribute name	Type	Unit	Description
	Boolean		
	Float		

Events

Event name	Description

Format to propose a new vendor-specific function

Thanks to its flexible device data model, TALQ can already be used to provide interoperability between Central Management Software and suppliers' outdoor device networks in quite some smart city vertical markets, as well as allow them to cover multiple smart city applications within a single platform.

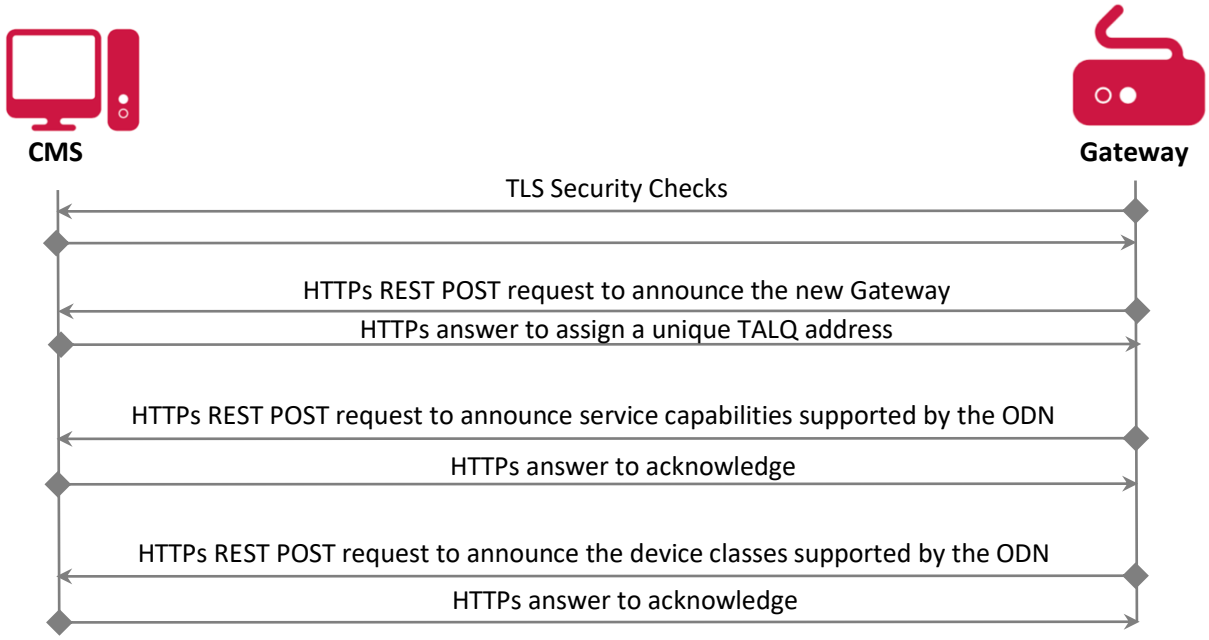


1.4 TALQ, a RESTful and JSON architecture

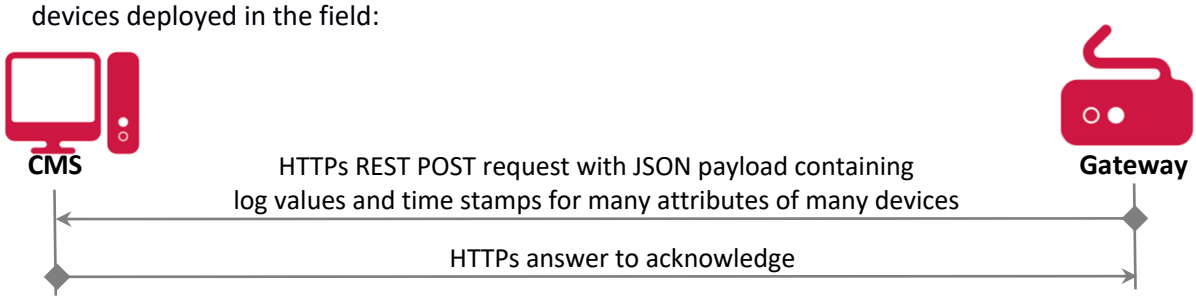
The TALQ 2.0 Smart City Protocol is based on a RESTful architecture and a JSON data model representation, because of their inherent simplicity of implementation. The protocol is described in this technical specification document as well as three Open API v3 (OAS 3) files which describe both the RESTful CMS and Gateway APIs and the JSON data model. Software developers can easily generate documentation or their source code using Open API tools (e.g. Swagger) and support TALQ in their CMS or Gateway.

The figure below illustrates some typical exchanges between CMS and Gateway as examples of TALQ protocol operation:

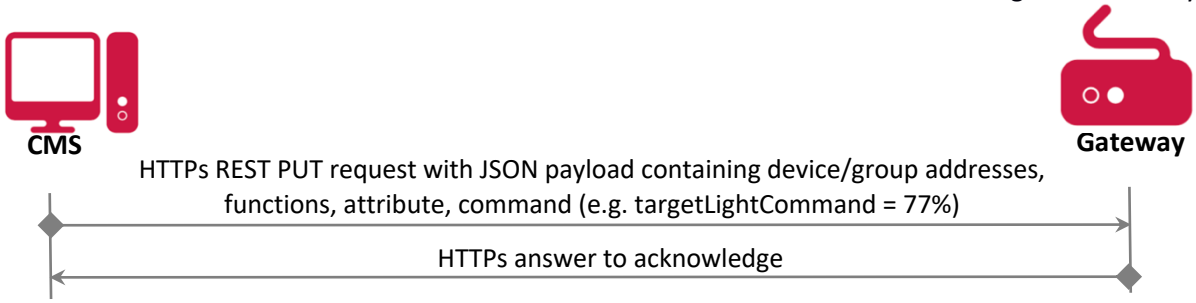
- The Bootstrap & Discovery process that enables a new Gateway to be connected to a CMS



- The Data Collect Service that allows log data to be sent to the CMS from one or more or a group of devices deployed in the field:

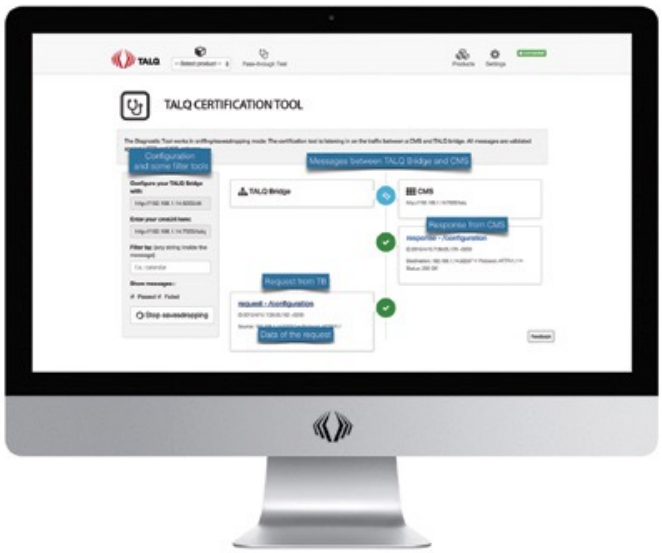


- The override control that allows an end-user to send a command to a device through the Gateway.



1.5 The TALQ Certification Program

The TALQ Consortium has set up a Certification Program with transparent procedures and a TALQ Test Tool to ensure full interoperability between a TALQ compatible CMS and a TALQ compatible Gateway. TALQ Members can challenge their own CMS or Gateway with the TALQ Test Tool until their implementation of the TALQ Smart City Protocol is successful. TALQ plugfest sessions allow the members to test their solution in collaboration with solutions from other vendors.



Official certification of TALQ compliance is awarded by the TALQ Certification Workgroup shortly after companies submit all of the necessary test result files and declarations for certification.

Certified TALQ compliant products are identified by the TALQ symbol and listed on the TALQ website.

2 DEFINITIONS AND CONVENTIONS

2.1 Definitions

Many words and phrases which carry a special meaning in the context of this document are defined in this section:

- **Central Management Software (CMS)**: An application which communicates with ODNs to enable remote configuration, control, command and monitoring of smart city connected devices.
- **Outdoor Device Network (ODN)**: A network of connected physical devices (e.g. Light Point Controllers, Waste Container Sensors, Parking Place Detectors, Air Pollution Sensors, etc.).
- **Gateway**: An application that provides access to ODN devices using the TALQ Smart City Protocol.
- **Device:** A logical representation of a physical device that is controlled and managed by the CMS using the TALQ Smart City Protocol. A device implements a set of functions and is uniquely identifiable within the ODN. The vendor is free to define its device with TALQ functions. The vendor may use more than one device to describe a physical device or gather multiple physical devices within one single device.
- **Function**: A group of attributes and events that describes particular functionality within a device. Some function types can be instantiated more than once in the same device.
- **Attributes**:
    - **Configuration attributes** are attributes that describe a device's capabilities and characteristics. They include configuration parameters that determine how specific features are implemented or executed in the device. For instance, temperatureHighThreshold in a temperature sensor will determine when the temperature sensor will trigger a temperatureTooHigh event.
    - **Operational attributes** are attributes that can be used to remotely command the device. The targetLightCommand (used to command a lamp to a particular value) in a lamp actuator is a typical example of an operational attribute.
    - **Metering attributes** provide performance data about specific functions in the device. Energy, active power, supply voltage and current are examples of metering attributes that may be available in functions such as an electrical meter and a lamp monitor.
    - **Status attributes** can be associated with some TALQ events and may be used to retrieve the result of the preceding event. Implementation of status attributes is optional. To avoid duplication these attributes are not explicitly described in this document, but if implemented they shall be of Boolean type and have the same name as the corresponding type of event.
- **Events** are messages that are generated by a Gateway to notify a CMS about an event such as a failure or a change. temperatureTooHigh is a typical event used to inform the CMS that the temperature metering attribute of a temperatureSensor function in a device is above its temperatureHighThreshold configuration attribute.

2.2 Abbreviations

This section provides a list of acronyms and abbreviations used in this document:

- **CMS**: Central Management Software
- **MAC**: Message Authentication Code
- **NTP**: Network Time Protocol
- **OID**: Object Identifier
- **ODN**: Outdoor Device Network
- **HTTP**: Hypertext Transfer Protocol
- **TCP**: Transmission Control Protocol
- **TLS**: Transport Layer Security
- **UDP**: User Datagram Protocol
- **URI**: Uniform Resource Identifier
- **UUID**: Universally Unique IDentifier
- **URL**: Uniform Resource Locator
- **JSON**: JavaScript Object Notation
- **RSA**: Public-key cryptography algorithm that is based on the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977 [RSA78].

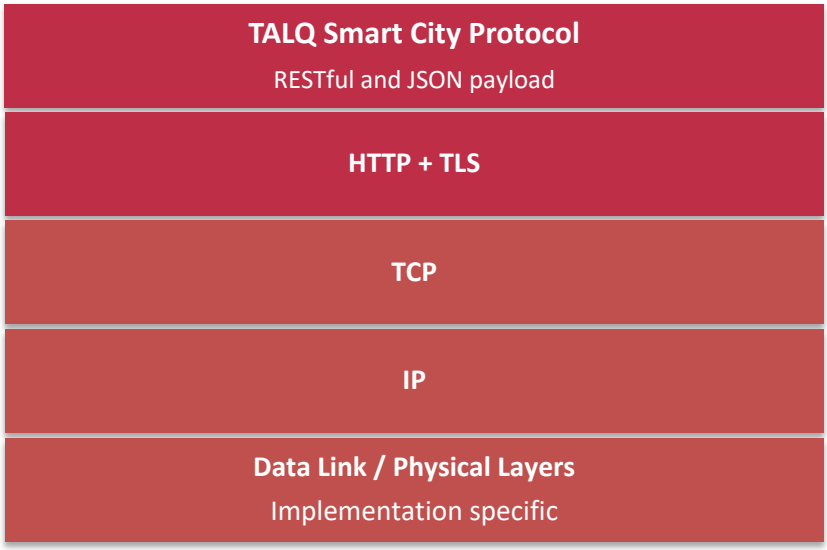
2.3 Conventions

MUST: This word, or the terms "***REQUIRED***" or "***SHALL***", mean that the definition is an absolute requirement of the specification.
MUST NOT: This phrase, or the phrase "***SHALL NOT***", mean that the definition is an absolute prohibition of the specification.
SHOULD: This word, or the adjective "***RECOMMENDED***", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT: This phrase, or the phrase "***NOT RECOMMENDED***" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
MAY: This word, or the adjective "***OPTIONAL***", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option ***MUST*** be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option ***MUST*** be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).
Mandatory (**M**) and ***Optional*** (**O**) features (attributes, events and services) are defined for the CMS and for the Gateway. Mandatory attributes, events and services shall be used to implement the functionality as specified, although how this data is used is outside the scope of the specification. Both CMS and Gateway shall accept optional attributes and vendor specific attributes/events (i.e. not trigger an error when receives such attributes/events) but they are not required to implement corresponding functionality.

3 TALQ ARCHITECTURE

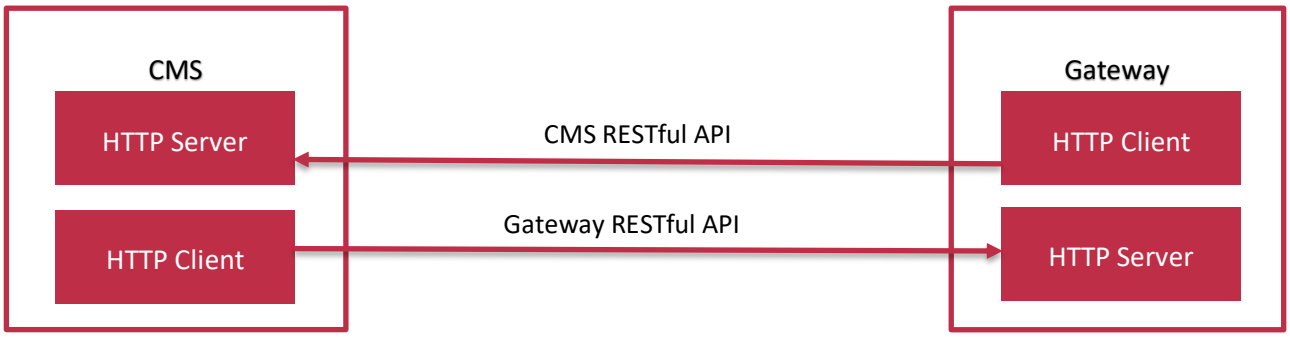
3.1 RESTful architecture

The TALQ Smart City Protocol is a bidirectional protocol based on RESTful architecture with JSON data payload that relies on underlying standardized data transport, security and network services (HTTP, TLS, TCP and IP) to establish communication between CMS and Gateways, as described below:



3.2 Bidirectional communication

REST is a common way to structure Web Service APIs. But REST is unidirectional while many Smart City markets require a bidirectional communication pipeline to allow remote control/command, device configuration, device monitoring and other data exchange. For instance, a CMS may need to send a command to turn on a light while a sensor may need to send a temperature reading to a CMS. A TALQ protocol implementation shall support bidirectional communication by providing an HTTP server at both ends of the communication pipeline as described below.



TALQ RESTful implementation to provide bidirectional communication

3.3 Address of the Gateway and the CMS

A CMS may configure, control, command and monitor many devices through many Gateways. On the other hand, a Gateway may also interface to more than one CMS. For the purpose of uniquely identifying the demanding party (CMS or Gateway), clients must include a REST parameter named "clientAddress" in each REST request. The “clientAddress” shall be the TALQ Address of the entity sending the REST request (Gateway or CMS). Below is an example of a properly formatted REST request where the Gateway’s TALQ address is “133a8753-e34c-332p5-b543-577755648453”:

```
Request URL:
https://<cmsUri>/devices/<deviceAddress>/temperatureSensor/temperature?clientAddress=133a8753-e34c-332p5-b543-577755648453
Request Method: GET
Status Code: "200 OK"
Request Header: {
  “talq-api-version”: “2.4.0”
}
```

The Gateway address shall be created and assigned by the CMS as the first step of the bootstrap process (refer to section 5.1 for more details of the bootstrap process). The CMS and Gateway addresses must be Universally Unique IDentifiers (UUID). The method through which the CMS determines its own TALQ address is not in the scope of TALQ.

3.4 TALQ RESTful API

The address of any resource presented by the CMS and Gateways shall use the standard URI syntax specific to HTTP/1.1 as per [RFC 1630], [RFC 1738], and [RFC 1808]. Formally defined and registered URN namespace definitions are recommended to address the application level resources as referenced in [RFC 2717], and [RFC 1808].
To support bidirectional communication, the CMS and the Gateway shall both run a HTTP Server and HTTP Client, providing a RESTful API to the other party. The “cmsUri” (base URI used to communicate with the CMS) and “gatewayUri” (base URI used to communicate with the Gateway) should always end with a slash “/” although both endpoints must accept URIs with or without it. All other URIs are relative.
The meaning of the HTTP REST verbs is defined as follows:

HTTP verb	Meaning
GET	Fetches the representation of a resource, without modifying it.
POST	Creates a new resource from a provided representation of the resource
PUT	Replaces the entire existing device definition. For the function array this will mean the existing device functions will be replaced with those specified in this call.
PATCH	Partially updates the existing device definition. For the function array each element passed in will be cross-referenced against items in the existing function array using the id. If they match, the existing function will be updated. If there is no match the function will be added. To remove functions from the array, see ‘PUT’, where an empty or reduced function array can be passed in.
DELETE	Deletes the resource from the system.

Here are some examples of the TALQ URI structure to get, create, update and delete resources by the Gateway on the CMS or by the CMS on the Gateway:

Objective	RESTful API (URL)
Get the list of all the devices that exist in the Gateway, their functions and attributes	HTTP GET https://<gatewayUri>/devices?clientAddress=<cmsAddress> The answer shall be a HTTP Response Code 200 with a JSON payload providing the list of devices, functions, attributes.
Get the list of functions and attributes for one particular device in the Gateway	HTTP GET https://<gatewayUri>/devices/<deviceAddress>?clientAddress=<cmsAddress> The answer shall be a HTTP Response Code 200 with a JSON payload providing the list of functions and attributes from this device.
Set the value of a list of configuration attributes of a device on a Gateway	HTTP PATCH https://<gatewayUri>/devices/<deviceAddress>?clientAddress=<cmsAddress> associated with a JSON payload to list the values to set. The answer shall be a HTTP Response Code 202. *Note: in the case of a function array, each array element passed in will be used to update any existing element with the same function id. If there is no existing element with the same function id, one will be added.*
Report a list of log values from one device by a Gateway to the CMS	HTTP POST https://<cmsUri>/log-report?clientAddress=<gatewayAddress> associated with a JSON payload to list the log values to report. The answer shall be a HTTP Response Code 201. or HTTP POST https://<cmsUri>/devices?clientAddress=<gatewayAddress> associated with a JSON payload to list the log values if they relate only to devices' attributes and events. The answer shall be a HTTP Response Code 201.
Read a particular metering attribute (supplyVoltage) on one device by the CMS	HTTP GET https://<gatewayUri>/devices/<deviceAddress>/<lampMonitorAddress>/supplyVoltage?clientAddress=<cmsAddress> The answer shall be a HTTP Response Code 200 with a JSON payload providing the attribute value.
Set the value of one particular attribute of a device in the Gateway	HTTP PUT https://<gatewayUri>/devices/<deviceAddress>/<lampActuatorAddress>/lampTypeId?clientAddress=<cmsAddress> associated with a JSON payload to set the new value of lampTypeId attribute. The answer shall be a HTTP Response Code 201. *Note: PUT may be used to remove functions from a function array.*
Create a list of devices in the CMS	HTTP POST https://<cmsUri>/devices?clientAddress=<gatewayAddress> associated with a JSON payload to provide the device list and their functions/attributes. The answer shall be a HTTP Response Code 201.
Delete a device in the CMS	HTTP DELETE https://<cmsUri>/devices/<deviceAddress>?clientAddress=<gatewayAddress> The answer shall be a HTTP Response Code 200.
Add a list of new calendars in a Gateway	HTTP POST https://<gatewayUri>/calendars?clientAddress=<cmsAddress> associated with a JSON payload to list the content of the calendars. The answer shall be a HTTP Response Code 201.

When a RESTful request includes a list of objects or any other entity, and one of them triggers an error (e.g. it does not exist) , the whole request is rejected. For example, a PUT/devices with a list of devices, where one of them does not exist at the target endpoint, shall be rejected with a HTTP error code 404.
The default TALQ namespace URI is http://talq-consortium.org/talq.json. The reference is http://json-schema.org/specification.html
The following sections provide examples of RESTful API and associated JSON payload to perform some of the possible TALQ actions between a CMS and a Gateway.
The detailed RESTful API and associated JSON schema are available in three OpenAPI v3 files published in the members' area of the TALQ Consortium website:

- *[date]-talq-api-cms-[MAJOR-MINOR-PATCH].json* describes the RESTful API provided by the CMS
- *[date]-talq-api-gateway-[MAJOR-MINOR-PATCH].json* describes the RESTful API provided by the Gateway.
- *[date]-talq-data-model-[MAJOR-MINOR-PATCH].json* describes all object types that are used by both RESTful APIs, CMS and Gateway.

A TALQ Protocol implementation shall include all mandatory features described in those files and may include one or more optional features.
Given a version number MAJOR-MINOR-PATCH, the:

- MAJOR version increments when incompatible API changes have been made,
- MINOR version increments when new functionality in a backwards-compatible manner has been added, and
- PATCH version increments when backwards-compatible bug fixes have been made.

The functions and attributes that compose devices are listed in detail in the OAS files. In case of any incompatibility or ambiguity between this document and the above listed OpenAPI files, the OpenAPI files shall prevail.

3.5 HTTP options and constraints

This Section describes how HTTP shall be used in the TALQ Smart City Protocol. **Version**

The use of HTTP version 1.1 [RFC2616] is mandatory. It enables reuse of the underlying transport connection for multiple requests.

3.5.2 HTTP Headers

This section describes the HTTP headers that shall be supported in a TALQ Smart City Protocol implementation. Other headers not mentioned in this specification may be used for other purposes such as security.

Topic	Comment
host	Mandatory in HTTP 1.1. It is used for hosting multiple domains on the same IP address.
content-length	All entities (requests and responses) shall have an explicit Content-Length header. This implicitly prevents use of the chunked transfer encoding.
content-type	All JSON entities (requests and responses) shall have a Content-Type header with value "application/json". "UTF-8" shall be the charset to be used in all TALQ data exchanges.
content-encoding	The following are permitted for all entities (requests and responses):

	• No Content-Encoding header • Content-Encoding: deflate • Content-Encoding: gzip No other content-ending may be used. The three encoding options listed above are mandatory for the server (Gateway or CMS). If a request uses a content encoding that is not supported by the server, the server shall return a response code 415 (Unsupported Media Type). See Section 14.11 of [RFC2616]. Note that this application is slightly different from how a typical web browser would work. A web browser would send an Accept-Encoding header in the request with the methods it can support, and the server has the choice to send the response back compressed or uncompressed. As the TALQ Smart City Protocol is designed to transmit large amounts of data, it is necessary to support a way to also compress those requests. Thus, the server shall always answer with the encoding option indicated in the client request.
accept-encoding	This header may be used in the Data Package Transfer Service to indicate the encoding option accepted by the Gateway, which can be either gzip or deflate.
connection	The TCP connection is kept open by default with HTTP 1.1. The connection header may be used in either a request or response to indicate that the underlying transport connection shall not be reused after the current request is executed. See Section 14.10 of [RFC2616].
cache-control	In case HTTP is used without TLS (see section 3.6) there may be HTTP proxies between the CMS and the Gateway. Requests are not cached but can be transformed by a proxy. Responses can be cached and transformed. A cache-control header equal to “no-transform” shall be added to both requests and responses.
resync	This header may be used by the Gateway in the Data Package Transfer Service to request a part of a package to recover from an aborted download. The unit type used for the range header shall be “bytes”.
content-range	This header shall be used by the CMS in the Data Package Transfer Service to respond to a Gateway request including the range header. The CMS shall always include a content-range header reflecting the data sent and always including an explicit instance length. The unit type shall be “bytes”.
talq-api-version	This header shall be used by the CMS and Gateway to identify the supported version of TALQ REST requests.

3.5.3 HTTP Response Codes

All use of HTTP response codes shall be compliant with the standard [RFC 2616]:

Code	Message	Typical Usage
200	OK	GET - DELETE
201	Created	POST – PUT
202	Accepted	POST – PUT – PATCH
204	No content	GET – PUT – PATCH
206	Partial content	GET
301	Moved Permanently	GET – POST – PUT - PATCH
307	Temporary Redirect	GET – POST – PUT - PATCH
400	Bad Request	POST – GET – PUT – PATCH – DELETE
401	Unauthorized	POST – GET – PUT - PATCH – DELETE
403	Forbidden	POST – GET – PUT - PATCH – DELETE
404	Not found	GET – PUT – PATCH – DELETE
409	Conflict	POST – PUT

415	Unsupported Media Type	POST – GET – PUT – PATCH – DELETE
416	Requested Range Not Satisfiable	POST
429	Too Many Requests	POST – GET – PUT – PATCH – DELETE
5xx	Unknown Error	POST – GET – PUT – PATCH – DELETE

All messages described in the HTTP RESTful standard may be used. Specific error messages appear in the OAS files.

3.6 Security

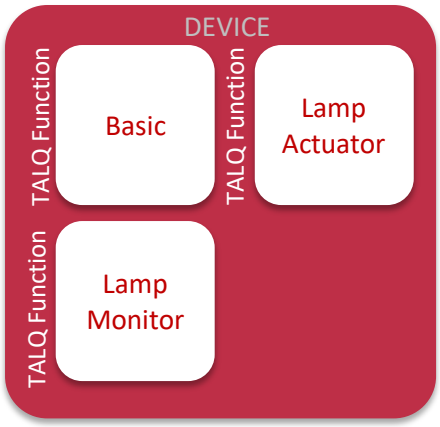
The selection of HTTP/1.1 as the default application data exchange protocol between Gateways and CMS leads to TLS for providing a secure channel for the communication. The TLS Handshake Protocol provides mutual authentication based on digital certificates derived from a trusted Certificate Authority or self-signed certificates, so that not only the server but also the client can be authenticated. The TLS Record Protocol provides encryption and message authentication of the application data to prevent eavesdropping, tampering or message forgery.
TLS support is mandatory in both CMS and Gateway. Vendors may be required to disable it to run tests and to support non-production environments.
All transactions between the Gateway and CMS shall be based on HTTP over TLS [RFC 2818] using TLS version 1.1 [RFC 4346] or later versions of TLS (e.g. version 1.2 [RFC 5246]) if implemented in both CMS and Gateway.
The TALQ Specification does not specify how the trust infrastructure is defined or how that trust infrastructure is established in installed systems. This is the responsibility of the entities that deliver the operational system.
Security within the ODN is outside the scope of the TALQ Specification and is the responsibility of the ODN supplier.
As described in section 4.1, an implementation of TALQ may control access to certain entities if multiple CMS may be connected; the indicated clientAddress may be used, or (for example) a vendor may choose to employ a token / API key in an HTTP header of TALQ messages.

4 TALQ DATA MODEL

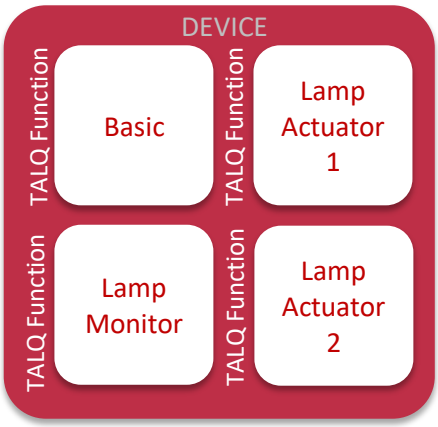
4.1 TALQ devices

The TALQ data model enables vendors of smart city connected physical devices to describe their devices using TALQ functions, each function having attributes and events that can be read (GET), sent or set (PUT, PATCH, POST) by the Gateway or the CMS. TALQ enables any CMS connected to a Gateway to have such access to all functions and devices. However, a TALQ implementation shall prevent access to a Gateway Function related to the connection of a different CMS. A TALQ Gateway may also prevent or limit some other access, for example limit control of certain actuators to a subset of connected CMS. Any measures to limit access are outside the scope of TALQ, see section 3.6.
TALQ functions, and their attributes and events, are defined by the TALQ Consortium and described in detail in the OAS files associated to this document. When using a TALQ function to describe a device the vendor shall implement each mandatory attribute and event of this TALQ function and may implement one or more of the optional attributes and events.
Here are some examples of possible devices and the functions that could compose them:

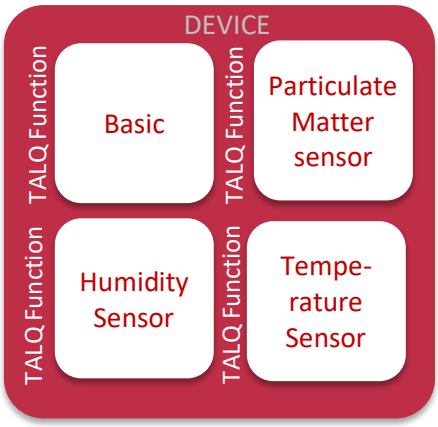
Light Point Controller from Vendor A



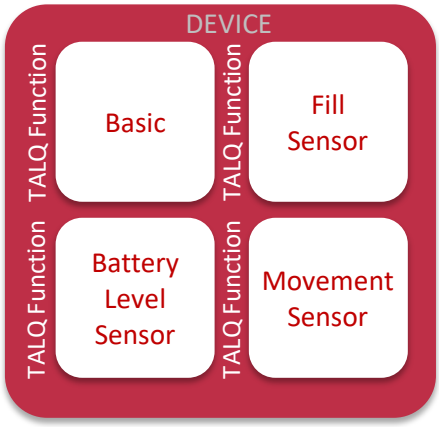
Light Point Controller from Vendor B

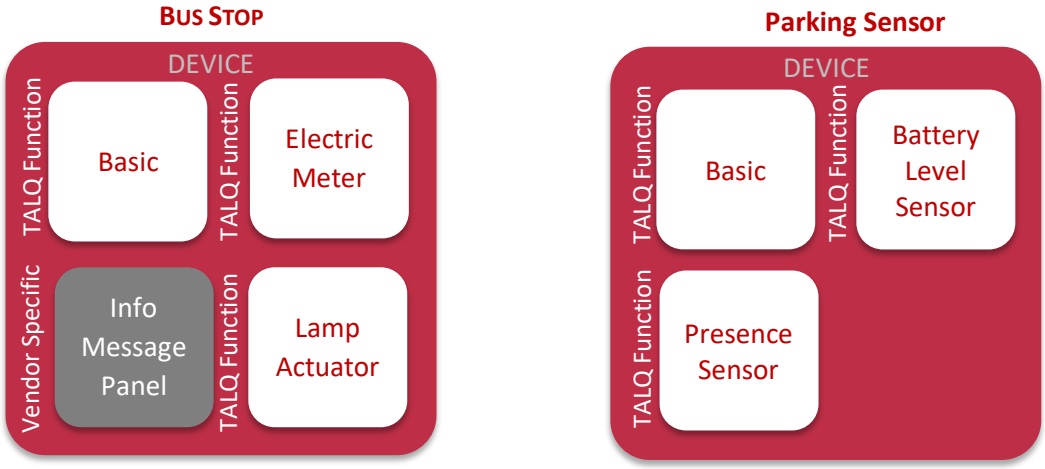


Environment Sensor



Waste Monitoring Sensor





The Gateway supplier shall define its own device classes following the format and constraints defined by the TALQ Smart City Protocol. The definition of device classes is used in the configuration service as a way to announce device capabilities (i.e. supported functions, attributes and events) with minimal data transfer overhead. Once a device class is announced, other device instances can be announced without repeating the complete device description.

When a new class of device is defined by a vendor to best describe its device, it should, where possible, comprise TALQ-defined functions to maximize interoperability. As described in section 5, TALQ-defined functions and services already support many Smart City use cases but when an implementation requires a function which is not already specifically defined in TALQ, a TALQ-defined generic function (e.g. Generic Sensor, Binary Sensor, Generic Actuator) can be used if appropriate. This may require communication between the CMS vendor and the ODN vendor to ensure the generic data returned can be appropriately interpreted and displayed in the CMS.

Alternatively, a vendor-specific function can be defined, although in general a CMS would not be aware of the behavior of vendor-specific functions. Again, communication between the vendors of any CMS and ODN which are expected to support the function may be required to establish a common understanding of the semantics of the function. Such a function should if possible use existing TALQ-defined Attributes and Events.

TALQ members developing vendor-specific functions are invited to propose these functions with associated attributes and events to the TALQ Consortium, to be considered for inclusion in a future version of the TALQ Specification through the process defined on the TALQ Consortium official web site.

4.2 TALQ functions

A TALQ function is composed of configuration attributes, operational attributes, metering attributes and events (the events correspond to status attributes). The configuration, control, command and monitoring of that function can be achieved by sending a RESTful request (GET, PUT, POST, DELETE, PATCH) to manipulate these attributes and events, or by using other more complex TALQ resources (e.g. override, log reports, control programs, etc.).

The complete and detailed descriptions of these TALQ functions, their attributes and events are described in detail in the OAS file. The list below is given here as a set of examples:

Function	Description	Examples of attributes (refer to the TALQ web site to get the accurate and updated list of TALQ functions, Attributes and Events)
Basic	The Basic function describes the properties of the physical device, such as its identifier, its geographical position and more. This function also provides information on time reference such as device time zone and local time. The ntpServer and a ntpSynchPeriod attributes are optional and may be used for time synchronization. This function is mandatory in any physical device for both sides, Gateway and CMS.	• assetId • displayName • longitude • latitude • serial • timeZone • ntpServer • ntpSyncPeriod • currentTime • … • installationDateTime
Gateway	The Gateway function includes the attributes to enable the communication between a CMS and the Gateway according to the TALQ Smart City Protocol. In a given ODN there shall be one and only one instance of the Gateway function per supported CMS connection.	• gatewayAddress • gatewayUri • cmsURI • vendor • … • retryPeriod
Communication	The Communication function hosts attributes related to the communication within the ODN, i.e. between ODN devices and the Gateway. Although communication within the ODN is not in the scope of the TALQ Smart City Protocol, this function may enable the CMS to configure some parameters in the ODN and/or to display network performance.	• communicationType • transmitPower • repeatingEnabled • numberOfHops • signalStrength • … • communicationFailure • applicationType
Lamp actuator	The Lamp Actuator function includes attributes related to switching and dimming one or more lamps with one single actuator. The Lamp Actuator function refers to a Lamp Type, which is a lighting-specific resource that gathers parameters that are typical to that lamp, such as the lamp wattage, the electric threshold for that lamp or the control mechanism to switch and dim this lamp (e.g. DALI, 1-10V, other). Refer to the JSON schema for a complete description of the lamp type parameters.	• targetLightCommand • calendarID • lampTypeID • feedbackLightCommand • actualLightState • invalidCalendar • invalidControlProgram • applicationType
Lamp monitor	The Lamp Monitor function allows monitoring of lamp parameters such as electrical parameters, energy consumption, temperature and more. The Lamp Monitor also refers to Lamp Types to use some of the lamp type parameters such as lowCurrentThreshold, highCurrentThreshold, warmUpTime and to identify failures and events. Refer to the JSON schema for complete description of the lamp type parameters.	• supplyType • lampTypeID • operatingHours • temperature • supplyVoltage • supplyCurrent • activePower • powerFactor • activeEnergy • … • lampFailure • supplyCurrentTooHigh • … • applicationType
Electrical meter	The electrical meter function supports measurement such as voltage, current, power, energy, and power factor. This function may be associated with cabinet controllers, electrical	• totalPower • totalVA • supplyVoltage • phase1Voltage

	meters installed in switch boxes or any other device that measures energy. It can be used to measure either single phase and three phase circuits.	• phase2Voltage • phase3Voltage • … • totalPowerTooHigh • applicationType • And many others
Photocell	The Photocell function models the capabilities of a photocell that detects ambient light level. Uses may include lighting control.	• onLevel • offLevel • photocellOutput • applicationType
Light sensor	The Light Sensor function provides the output of a light sensor and allows Gateways to send events in case the value is above/below configurable thresholds.	• levelLowThreshold • levelHighThreshold • lightLevel • levelTooHigh • levelTooLow • applicationType
Temperature sensor	The Temperature Sensor function allows a CMS to monitor the temperature and send events in case the value is above/below configurable thresholds.	• temperatureHighThreshold • temperatureLowThreshold • fireDetectionThreshold • temperature • temperatureTooHigh • temperatureTooLow • fireDetected • applicationType • minMeasuredTemperature • maxMeasuredTemperature • measuredTemperatureSince
Humidity sensor	The Humidity Sensor function allows a CMS to monitor the humidity and send events in case the value is above/below configurable thresholds.	• humidityHighThreshold • humidity • humidityTooHigh • applicationType
Particulate Matter sensor	The Particulate Matter Sensor function allows a CMS to monitor PM10, PM2.5 and PM1 levels and send events in case the value is above/below configurable thresholds.	• pm1HighThreshold • pm2-5HighThreshold • pm10HighThreshold • pm1 • pm2-5 • pm10 • pm1TooHigh • pm2-5TooHigh • pm10TooHigh • applicationType
Presence sensor	The Presence Sensor function allows a CMS to detect presence. This function may be used in Parking Place detectors as well as in dynamic outdoor lighting scenario.	• presenceStatus • presenceStatusChanged • applicationType
Movement sensor	The Movement Sensor function allows a CMS to detect physical movement of a device. This function may be used in Waste Container sensor to detect that container has apparently been emptied or is not in the proper position, as well as in asset tracking applications. **DEPRECATED by 2.2.0**: This function has been deprecated and it will be removed in the next MAJOR release. Please use the new LocationSensorFunction instead.	• movementThreshold • movementDetected • notInProperPosition • applicationType
Filling level sensor	The Filling Level Sensor function allows to measure how full a container is and send an	• levelHighThreshold • containerHeight • containerVolume

	event in case the value is above configurable thresholds.	• fillingHeight • fillingPercentage • containerFull • contentsType • contentsOtherType • applicationType
Battery level sensor	The Battery Level Sensor function allows to measure the charge of the battery, monitor the battery, identify whether the associated device is using the battery and send events in case the value is below configurable thresholds.	• batteryLevelLowThreshold • batteryLevel • batteryLevelTooLow • powerSource • applicationType
Generic actuator	The Generic Actuator function enables to control an actuator, for instance in an electrical cabinet or an irrigation valve.	• targetCommand • feedbackCommand • defaultState • actualState • applicationType
Generic sensor	A Generic Sensor function can be used to model any sensor that provides an analog output and send events in case the value is above/below configurable thresholds.	• levelHighThreshold • levelLowThreshold • level • levelTooHigh • levelTooLow • applicationType
Binary sensor	A Binary Sensor function can be used to model any sensor that provides a binary output.	• level • sensorOutputOn • applicationType
Solar Battery Charger	A solar battery charger is used to charge a battery with solar energy. Typical use cases are energy demanding off-grid applications like solar lighting, solar vehicle charging (cars and bikes), public transit information, traffic control, public security (CCTV) and many more. The aim of this function is to provide very general attributes that allow the CMS to configure and monitor such applications.	• inputVoltage • inputCurrent • outputVoltage • outputCurrent • chargerTemperature • PVTemperature • accumulatedEnergy • startChargeInputVoltage • endChargeInputVoltage • … • lowTemperature • highPower • charging • applicationType
Battery Management System	A battery management system is used to monitor the charging and discharging of a battery and protect the battery. Typical use cases are (off-grid) applications like solar lighting, solar vehicle charging (cars and bikes), public transit information, traffic control, public security (CCTV) and many more, where the battery is charged and discharged on a regular basis.	• batteryChemistry • nominalVoltage • nominalCapacity • … • overCurrentCharge • overCurrentDischarge • highTemperatureThreshold • highTemperature • applicationType
Traffic Counter	The Traffic Counter Function is used to provide statistics on the number of vehicles passing on the road. It allows to have the number of pedestrians, bicycles, cars or trucks for a certain period of time that is configurable by the CMS. It also allows to count the number of vehicles using diesel or petrol.	• roadUserNumber • accumulatedRoadUserNumber • roadUser • accumulatedSince • heavyTrafficDetected • heavyTrafficDetectedThreshold • trafficSamplingPeriod • applicationType
Location Sensor	The Location Sensor Function is used to indicate that an object has changed position attributes configurable by the CMS or based on internal	• expectedLocation • locationChangedThreshold • location

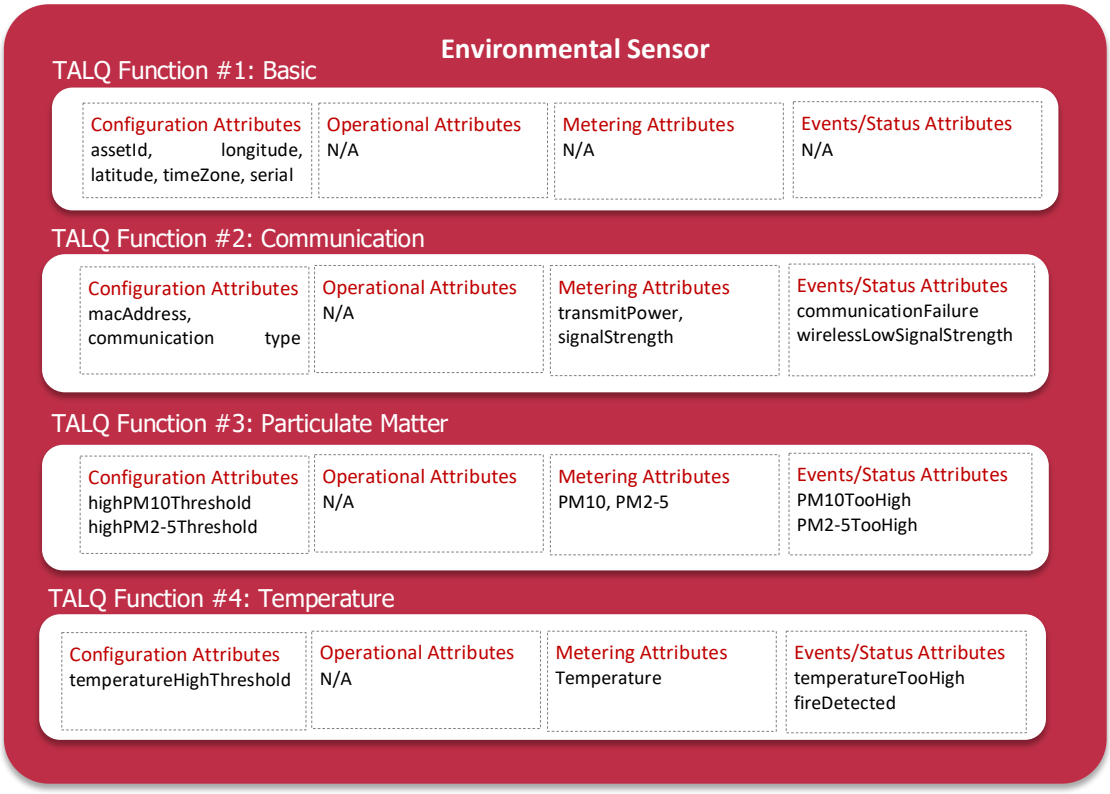
	setup of the vendor. For example, a specific location (latitude, longitude) of a device could be defined by the vendor. If the device is equipped with a GPS, it could send a specific event indicating that its position is different to the one defined by the CMS. We might also want to let the configuration to the vendor itself and simply define events notifying the CMS that the default configuration has changed. For example, a garbage bin could have its location defined based on a sensor placed on the floor. If the bin is not above this sensor, the vendor will trigger an event. In this last case, the CMS does not need to configure anything.	• locationChanged • uncertainty • compassDirection • velocity • speed • applicationType
Accelerometer	The Accelerometer Function is used to indicate that an object has had an impact with another object and to report its acceleration.	• impactDetectedAccelerationThreshold • accelerationSamplingPeriod • accelerationX • accelerationY • accelerationZ • acceleration • impactDetected • applicationType
Orientation	The Orientation function is used to indicate that an object has changed orientation based on attributes configurable by the CMS or based on internal setup of the vendor. The target orientation of the object could be configured by the CMS or could be handled by the vendor. In the latter case, the configuration is let to the vendor itself and events are triggered depending on internal configuration.	• expectedOrientation • orientationChangedThreshold • orientation • orientationChanged • applicationType
Fluid Level Sensor	The Fluid Level Sensor function allows to collect data and events about fluid levels. It could be used to measure fluid levels in channels, lakes, containers, etc.	• fluidLevelTooLowThreshold • distanceSensorBottom • fluidLevel • fluidLevelTooHigh • fluidLevelTooLow • applicationType
Waste Container	The Waste Container function allows to log when the container is collected and send events in case the date is above a configurable threshold. Additionally it sends events when the contents or container are tampered.	• lastCollectionDate • collectionLateThreshold • collectionLate • containerTampered • contentsTampered • wasteType • contentsOtherType • applicationType
pH Sensor	The pH Sensor allows to measure the pH and sends events if the value is above/below the configured thresholds.	• pH • pHHighLevelThreshold • pHLowLevelThreshold • pHTooHigh • pHTooLow • applicationType
Weight Sensor	The Weight Sensor allows to measure the weight and sends an event if the value is above/below the thresholds	• weight • weightHighThreshold • weightTooHigh • weightLowThreshold • weightTooLow • applicationType

Gas Sensor	The Gas Sensor function allows to measure the gas concentration and sends events if the level is above the configured thresholds.	• gasConcentration • gasHighConcentrationThreshold • gasConcentrationTooHigh • gasName • gasOtherName • applicationType
Simple Actuator	The Simple Actuator function includes attributes related to generic control and it represents the smallest unit for control purposes. This function allows the new profiles to use actuators without the complexity of calendars	• targetCommand • feedbackCommand • defaultState • actualState • applicationType
Time	The Time function provides information on time reference in the device time zone, and local time. This function is mandatory for both Gateway and CMS sides, although the x-talq-profiles definition will continue stating that it is optional until the release 3.0.0 to keep backwards compatibility. This will be modified at 3.0.0.	• timeZone • ntpServers • ntpSynchPeriod • currentTme • lastTimeSync • lastSyncError
Segment Monitor	The Segment Monitor function enables monitoring of segment parameters. Multiple segment monitor functions may be implemented by a single device.	• numberOfLoads • switchingErrorOn • switchingErrorOff • leakageDetected • cabinetDoorOpen • circuitBreakerTripped • localOverride • applicationType • segmentReference
Noise Monitoring Sensor	This sensor function enables monitoring basic noise data.	• noise • noiseHighThreshold • noiseTooHigh • applicationType • minMeasuredNoise • maxMeasuredNoise • measuredNoiseSince • abnormalNoiseDetected • typeOfNoise
Atmospheric Sensor	This sensor function enables monitoring basic atmospheric data such as barometric pressure, humidity, and temperature. This function complies with WMO standards as reported in the 'Guide to Instruments and Methods of Observation (WMO-No. 8) / Volume I - Measurement of Meteorological Variables'	• airTemperature • feelsLikeTemperature • relativeHumidity • dewPoint • atmosphericPressure • applicationType
Wind Sensor	This sensor function enables monitoring wind speed and direction. This function complies with WMO standards as reported in the 'Guide to Instruments and Methods of Observation (WMO-No. 8) / Volume I - Measurement of Meteorological Variables'	• windSpeed • windDirectionString • windDirection • windGust • windGustDirection • maxWindGust • maxWindGustSince • applicationType
Precipitation Sensor	This sensor function enables monitoring precipitation, defined as the liquid or solid products of the condensation of water vapour falling from clouds, in the form of rain, drizzle, snow, snow grains, snow pellets, hail and ice pellets; or falling from clear air in the form of diamond dust. This function complies with	• precipitationRate • accumulatedPrecipitation • accumulatedPrecipitationSince • applicationType

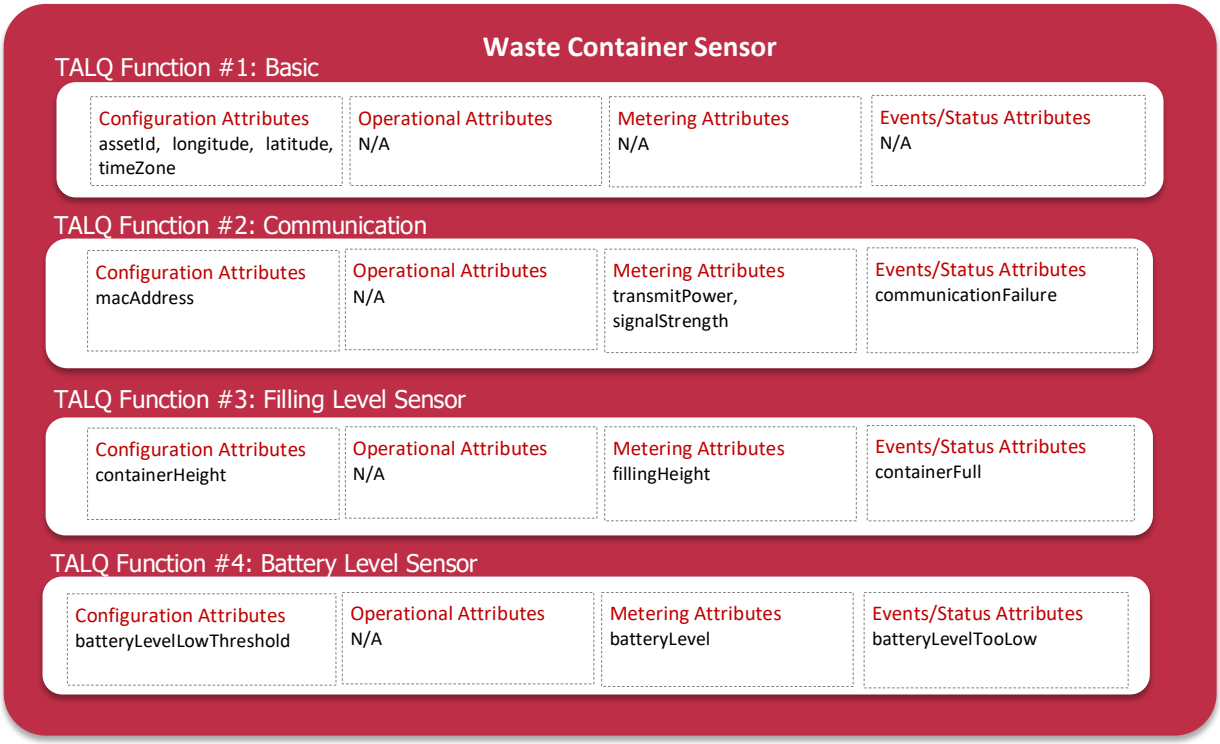
	WMO standards as reported in the 'Guide to Instruments and Methods of Observation (WMO-No. 8) / Volume I - Measurement of Meteorological Variables'	
Sky Sensor	This sensor function enables monitoring of other atmospheric phenomena. This function complies with WMO standards as reported in the 'Guide to Instruments and Methods of Observation (WMO-No. 8) / Volume I - Measurement of Meteorological Variables'	• cloudiness • solarDirectRadiation • visibility • applicationType
Gully Sensor	The Gully Sensor measures properties associated with street drains or gullies.	• overfull • levelWarning • grillOpened • siltLevel • applicationType
Water Flow Sensor	The water flow sensor function measures the water flow rate.	• flowRate • flowRateTooHighThreshold • flowRateTooLowThreshold • flowRateTooHigh • flowRateTooLow • maxFlowRate • minFlowRate • flowRateSince • applicationType
Water Quality Sensor	The water quality sensor function measures the quality of the water in the drinkable water distribution network, in water tanks or in lakes and rivers.	• pH • chlorine • orp • totalDissolvedGas • dissolvedOxygen • …. • NH4TooHigh • NH4TooHighThreshold • applicationType
Text Display Actuator	The Text Display Actuator is used to send text to a text-only or text mode graphics display within a PositionedTextState (text, xPos, yPos). Writing a string of text to the text resource causes it to be displayed at the selected X and Y locations on the display. If X or Y are set to a value greater than the size of the display, the position "wraps around" to the modulus of the setting and the display size. Likewise, if the text string overflows the display size, the text "wraps around" and displays on the next line down or, if the last line has been written, wraps around to the top of the display. Brightness and Contrast controls are provided to allow control of various display types including STN and DSTN type LCD character displays. Setting the clearDisplay to true causes the display to be erased.	• defaultState • actualState • targetCommand • feedbackCommand • stateChange • calendarID • invalidCalendar • invalidProgram • programChange • calendarChange • targetCommandChange • applicationType • maxXCoordinate • maxYCoordinate • level • contrast • clearDisplay
Parking Sensor	The parking sensor provides actual and accumulated occupancy duration as well as forbidden parking detection.	• occupancy • duration • accumulatedDuration • accumulatedSince • forbiddenParkingDetected • applicationType • sensorType

		• occupancyChangeToVacant • occupancyChangeToOccupied • overstayDetected • maxDuration
Parking Camera Sensor	The Parking Camera Sensor provides information about the parking slots that only computer vision can provide.	• totalSlots • slotsData • freeSlots • freeSlotIDs • averageDuration • accumulatedParkingSessions • accumulatedSince • forbiddenVehicleDetected • blockingVehicleDetected • badParkingDetected • applicationType • zoneReference

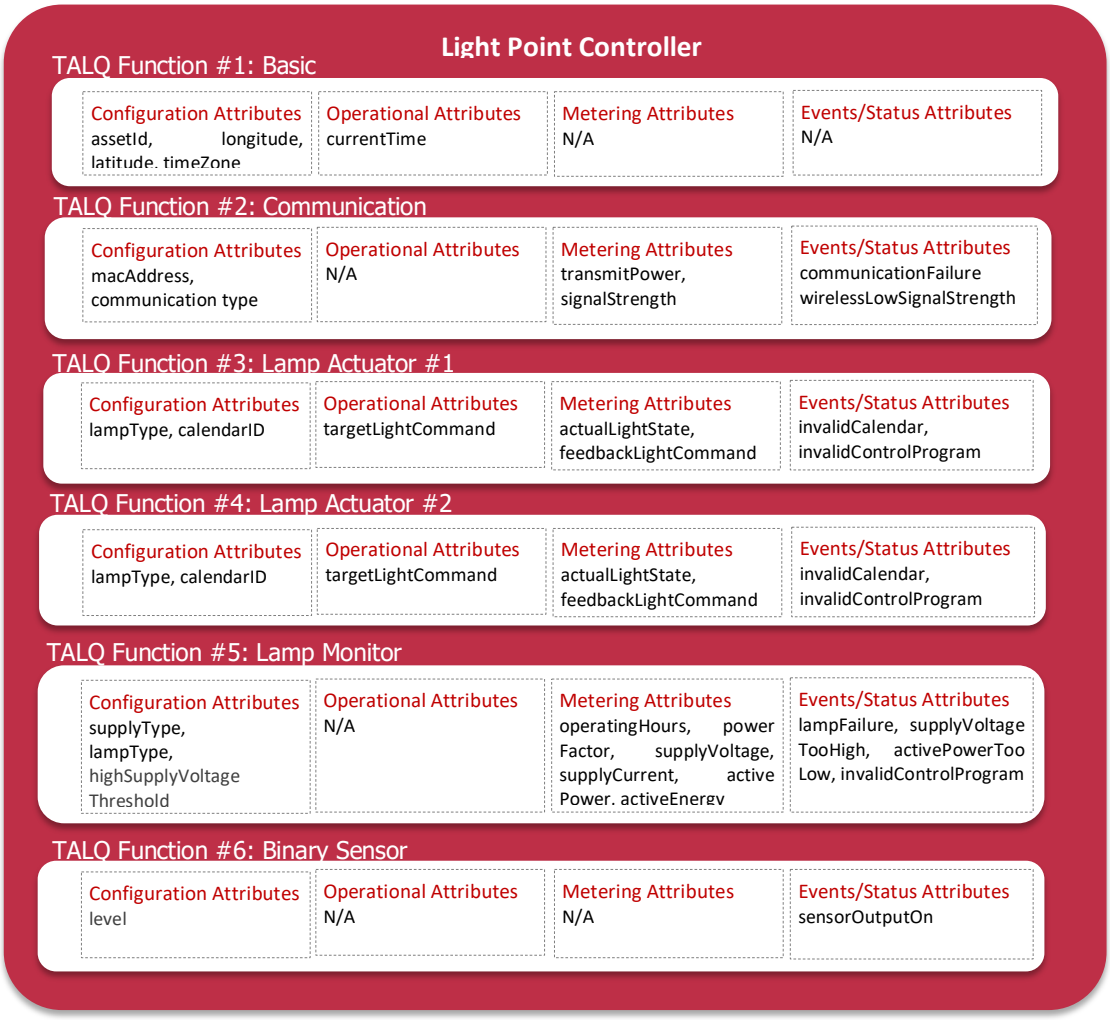
TALQ functions and device classes provide full flexibility to the vendor to model its devices. For example, an environmental sensor could be modelled by its vendor as follows:



The example below shows how a waste container sensor could be modelled with TALQ:



A Light Point Controller equipped with a motion detector to provide Dynamic Outdoor Lighting could be modelled by its vendor following the example below:



The above examples show the flexibility of the TALQ data model that allows vendor to have very different and creative devices, while allowing the CMS to understand the device capabilities and behavior and to configure, control, command and/or monitor it without the need to implement vendor specific features. The TALQ specification allow vendors to easily turn such a model into a device class that is announced by the Gateway to the CMS as part of the bootstrap process described in this document.

4.3 TALQ services

Each TALQ service defines the message exchanges and application behavior needed to manipulate function attributes in order to implement a certain set of features. Some services introduce other resources required to manipulate these function attributes. The following services are supported in TALQ.

4.3.1 Configuration service

The configuration service enables the Gateway to announce device and service capabilities (i.e. a list of supported services and of supported device classes including functions and attributes) and to announce devices to the CMS. It also enables the CMS to create devices in the Gateway and to configure their configuration attributes.
TALQ covers a variety of smart city uses cases, including street lighting. Some TALQ implementations may implement all possible TALQ functionality, but many will focus on a subset of applications.
Some TALQ functionality is mandatory across all applications. Some functionality is mandatory for certain applications but optional for others. Some functionality is simply optional across all applications.
To address this range of requirements, the use of "**Profiles"** in TALQ has been introduced. The existing indication of mandatory or optional for TALQ features has been adopted as a "Lighting Profile", and further profiles are defined for other applications. The mapping for each feature and each profile is contained in the data model OAS file.
The Gateway announces during the bootstrap process which profiles are supported. This information is important for the CMS in order to expect mandatory features based on the profiles announced.
The current version of TALQ provides the profiles below:

- Lighting
- Waste Management
- Environmental Monitoring
- Smart Traffic
- Smart Parking

The configuration service shall be supported by both CMS and Gateway in order to enable the other TALQ services necessary to operate and manage the system, such as on-demand read, control, group management and data collection.

4.3.2 On-demand read service

The on-demand read service enables the CMS to read any metering attribute from any function of any device. (This service is supported by a standard RESTful GET API on the device/function/attribute resource.)
The on-demand read service shall be supported by the Gateway.

4.3.3 Control service

The control service describes the mechanisms to operate commands in order to enable both schedule-based control and override control. The override control mode enables the CMS to actuate any command over a device or groups of devices as needed, e.g. open/close a valve to start/stop water irrigation, switch a lamp in real time, send a message to an information panel (note that such a basic actuator command can also be implemented with a simple PUT or PATCH to the related attribute). The schedule-based control enables the CMS to pre-configure the behavior of one command over one or more devices based on one or more factors, such as time and sensor inputs, e.g. open/close water irrigation valve when humidity is below a threshold, increase light level when a pedestrian is detected, switch ON a light at sunset.
The concepts of calendar and control program are introduced in the control service to achieve interoperability on the configuration and execution of schedule-based control.
The control service shall be supported by both CMS and Gateway.

4.3.4 Group management service

The group management service provides the mechanisms to define and manage groups. Groups can be used across TALQ services whenever there is a need to address a set of devices and/or functions together. For instance, an override control command may be sent to a group of devices, or the same control program may be distributed to a group of lamp actuators. A group may also include other groups as members. Groups can also be used in the configuration of the data collection features.
It should be noted that groups are used for communication and addressing purposes only. For instance, assigning a calendar to a group associates the calendar with that group at the moment of assignment. Later changes to the group do not assign the calendar to the new members nor do they remove the calendar from members removed from the group. The CMS shall manage such configuration changes independently.
The group management service shall be supported by the CMS but is optional for the ODN.

4.3.5 Data collection service

The data collection service enables the CMS to configure how an ODN reports metering attribute values, status information and events, and when or under what conditions the logged data is transferred to the CMS. This service enables adaptation of the data collection and transporting behavior to the needs of the specific CMS and ODN implementations. It also enables aggregated information to be reported to the CMS, which can result in more efficient use of bandwidth between CMS and Gateways. The data collection service covers scheduled data logging aspects by specifying a format to configure the recording and reporting of data and events.
The data collection service shall be supported by the Gateway.

4.3.6 Data package transfer service

The data package transfer service provides a mechanism to transfer data packages containing ODN vendor specific information, such as firmware updates, from the CMS to the Gateway. The CMS uses this service to forward data transparently to the Gateway. The content and format of the data transferred is outside the scope of the TALQ specification. It may include vendor specific information. One application of the data package transfer service is for the CMS to send firmware/software updates for end-devices or other vendor specific information.
The data package transfer service is optional for both CMS and Gateway.

4.4 TALQ resources

4.4.1 TALQ Addresses

TALQ addresses are used to identify devices, groups and other resources, and shall be unique within an ODN. TALQ addresses of Devices shall be as specified in RFC 4122 '**U**niversally **U**nique **ID**entifier (UUID) URN Namespace' (https://tools.ietf.org/html/rfc4122), and this format is also recommended for other TALQ addresses.

4.4.2 Resources

The URI associated with each RESTful API contains the type of resource involved in the RESTful API:

Type of resource	Description
devices	The devices composed of functions, themselves composed of attributes and events.
groups	The groups of devices.
device-classes	The description of the functions and attributes defined by the vendor to describe its physical devices.
control-programs	The set of rules to schedule the operations on a device automatically.
calendars	The set of calendar rules where control program applies.
logger-configs	The configuration parameters of a data logger

log-reports	A resource that is used by a Gateway to group multiple log values from many devices when sending them to the CMS
data-packages	A data package that is sent by the CMS to a Gateway, for example to update the firmware of a Gateway or of a set of devices.
Application-specific resources	**Description**
lamp-types	The set of parameters to describe lamp types for outdoor lighting network solutions.

A CMS shall not refer to a resource created by the CMS and expose this reference to a Gateway before the referred resource is at least sent once to the same Gateway.
A Gateway shall not refer to a resource created by the same Gateway and expose this reference to the CMS before the referred resource is at least sent once to the CMS.
Whenever the Gateway or the CMS encounters an invalid resource address, a TALQ invalidAddress event shall be generated using the log-report resource. Those resources created by a specific CMS (with the exception of devices and functions) are visible only to that CMS.
Vendor-specific attributes may be added in any TALQ entity. CMS and Gateway shall not raise an error if such an attribute is listed in a resource related request. Vendors having vendor-specific attributes should communicate with other parties to describe the format and explain how to use their vendor-specific resources.
A CMS shall not delete an entity referred by any other entity, and in such a case the request will be rejected by the Gateway.
A Gateway shall not delete an entity referred by any other entity, and in such a case the request will be rejected by the CMS.

5 TALQ USE CASES

5.1 TALQ bootstrap process

The bootstrap process is the process to establish the communication between a CMS and a Gateway using the TALQ Smart City Protocol. During the bootstrap process, the Gateway shall not send any information other than the information specified in this section and in the associated OAS files. The CMS shall reject any other request sent by the Gateway by answering with HTTP error code 403. The bootstrap process is composed of the following operations:

- **Security check pre-requisite**: The Gateway shall have been provisioned with the CMS's Uri ("cmsUri") and proper security keys (refer to Section 3.6 for security matters). The method through which the Gateway acquires the cmsUri is not part of the TALQ specification.
- **Step 1 - Gateway announcement**: The goal is for the Gateway to announce itself and its capabilities to the CMS and, in return, to receive a unique TALQ address that will be used in the data exchange.
- **Step 2 - Services announcement**: The Gateway sends the list of TALQ services, and their options, that it can support so that the CMS can align the way it will configure, control, command and monitor the Gateway's ODN.
- **Step 3 – Device class announcement**: The Gateway sends the list of device classes that it can support, together with their functions and attributes descriptions and constraints, so that the CMS align the way it will handle the devices on the Gateway's ODN.

If, prior to the current bootstrap, one or more other CMS were already attached then the Gateway shall signal an event(s) to pre-connected CMS indicating the attachment of another. Support for more than one CMS is optional.

5.1.1 Step 1: Gateway announcement

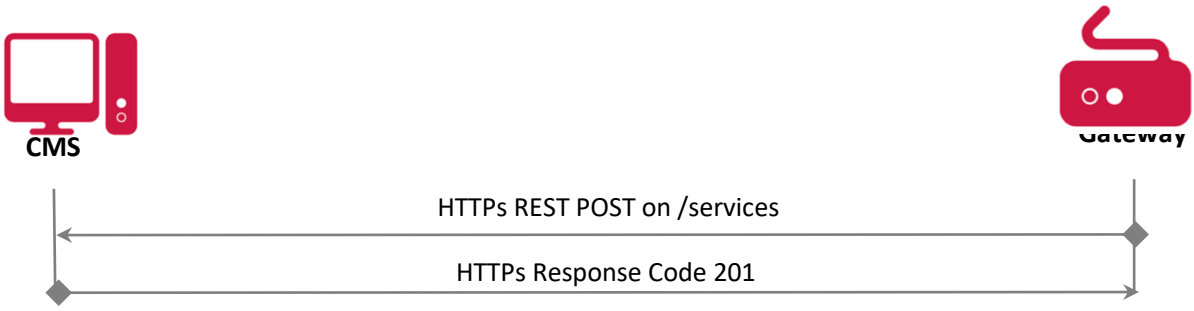
After a secure connection has been established, the Gateway shall send the following three REST requests to the CMS:

1. A POST request to /devices with a JSON Payload that provides the gatewayUri in its Gateway function. Although no device class has been announced yet, it is implicitly understood that this GatewayFunction is mandatory and thus is expected at the device representing the Gateway; and other functions can be included at this first step. When receiving such a request, the CMS shall create a unique gatewayAddress and send it, together with its cmsAddress, in the JSON payload of its response. Both addresses must be not NIL UUID.
2. A POST request to /device-classes?clientAddress=<gatewayAddress> to send the capabilities and the list of attributes that the Gateway device supports.
3. A PATCH request to /devices/<gatewayAddress>?clientAddress=<gatewayAddress> to send the value of each attribute of the device implementing the Gateway function. This request shall include the gatewayUri attribute inside the Gateway function so that the CMS knows where to send requests.

The final action of this step 1 is for the CMS to send a PATCH request to /devices/<gatewayAddress>?clientAddress=<cmsAddress> to set or update any configuration attribute in the device implementing the Gateway function. This shall include the TALQ address of the CMS.

5.1.2 Step 2: Services announcement

The Gateway shall then announce the list of TALQ services, and their options, that it can support, using a POST request to /services. The CMS shall respond with HTTP CREATED once the message is successfully processed. Here is an example of a request to send the list of TALQ services supported by the Gateway:

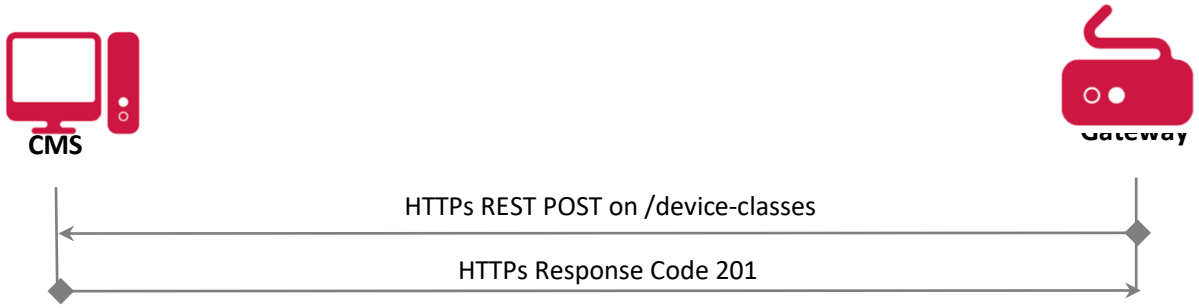


JSON Response

```
Request URL: https://<cmsUri>/services?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0",
}
Request Content: [
  {
    "name": "ControlService",
    "maximumCalendars": 100,
    "maximumPrograms": 300,
    "maxProgramsPerCalendar": 30,
    "maxSwitchPointsPerProgram": 10,
    "dayOffset": 0,
    …
  },
  {
    "name": "groupManagementService",
    "maximumNumberOfGroups": 20,
    "maximumGroupSize": 40
  },
  …
]
```

5.1.3 Step 3: Device class announcement

The Gateway shall then announce the list of device classes and associated functions and attributes it can support with its description and constraints, using a POST request to */device-classes*. The CMS shall respond with HTTP CREATED once the message is successfully processed. If the Gateway has no device class to announce to the CMS, it shall send an empty list of device classes inside the POST request so that the CMS can understand the bootstrap process has finished. Here is an example of a request to send the list of device classes supporting by the Gateway:



JSON Response

```
Request URL: https://<cmsUri>/device-classes?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  “talq-api-version”: “2.4.0”,
}
Request Content: [
  {
    "name": "<deviceClassName>"
    "functions": [
      {
         "name": "BasicFunction",
         "attributes":[
           {
             "name": "displayName"
           },
           {
             "name": "assetId"
           },
           …
         ]
      },
      {
         "name": "LampActuatorFunction",
         "attributes":[
           {
             "name": "lampTypeId"
           },
           {
             "name": "maintenanceFactor",
             "minValue": 0
             "maxValue": 100
           },
           {
             "name": "maintenancePeriod",
             "unit": "Hours"
           },
           …
         ]
      }
    ]
  },
  …
]
```

The Gateway may add functions and attributes to a device class after its announcement, but it shall never reduce the scope of a device class (e.g. remove an attribute) after announced. If the Gateway needs to reduce the scope of a device class, it shall announce it as a new device class to the CMS.

5.1.4 Resynchronizing a Gateway

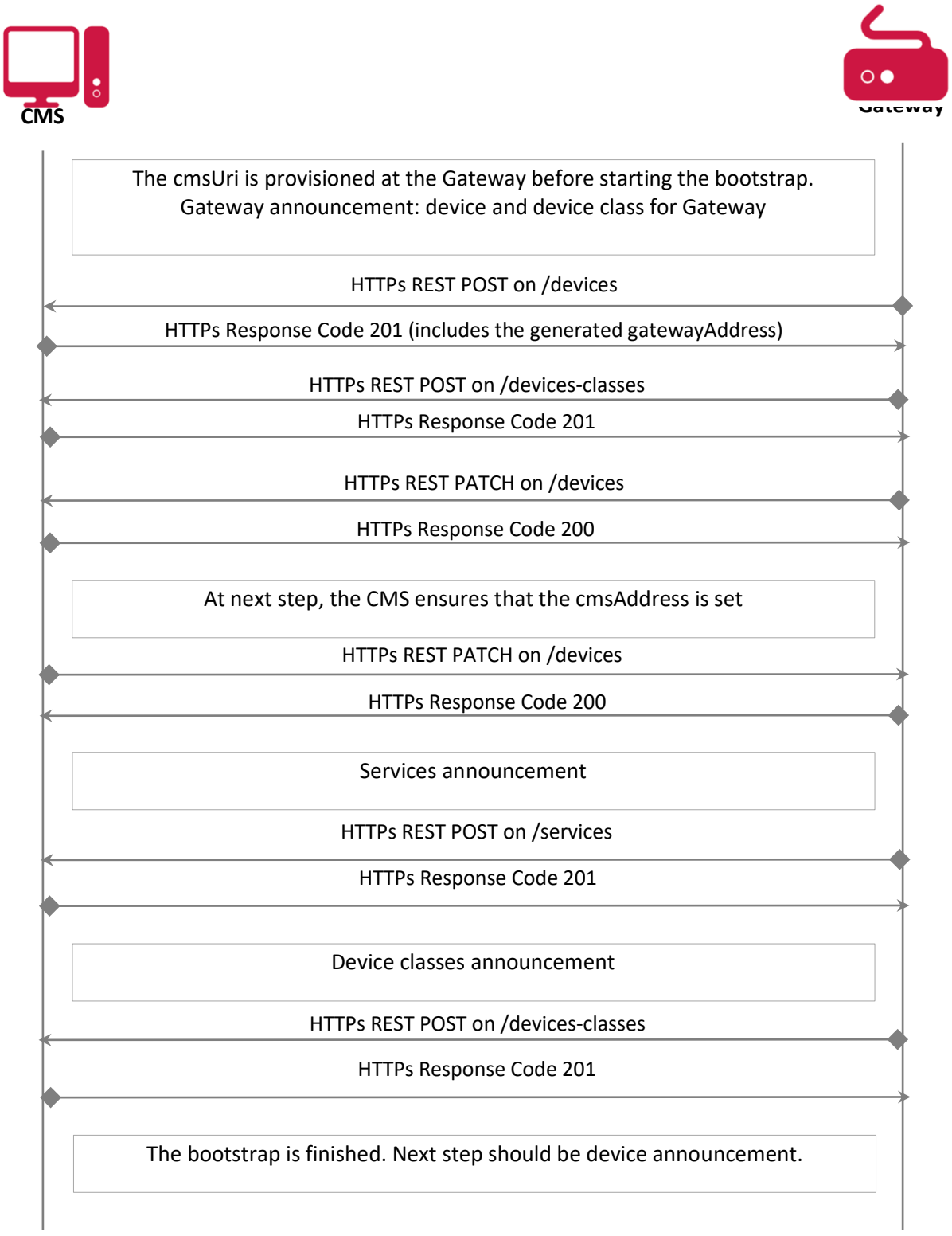
If the CMS needs the Gateway to resynchronize its address and all its data (e.g. remove all the device classes):

1. The CMS should send a DELETE request to https://<gatewayUri>/devices/<gatewayAddress>?clientAddress=<cmsAddress>
2. The Gateway shall remove all the data related to the CMS and send HTTP OK.
3. The Gateway shall then restart the bootstrap process as described above.

If the Gateway needs to resynchronize or to reset:

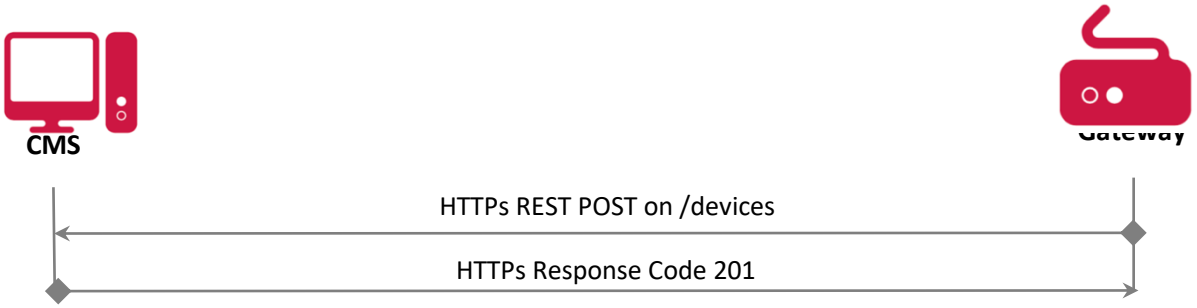
1. The Gateway shall send a DELETE request to https://<cmsUri>/devices/<gatewayAddress>?clientAddress=<gatewayAddress>
2. The CMS shall send a HTTP OK as a response after removing all the data attached to this Gateway.
3. The Gateway shall remove all CMS data related to this CMS.
4. The Gateway shall restart a bootstrap process.

As an example, see below the message exchanges between a Gateway and a CMS during the bootstrap process:



5.2 Announcing devices

Once the bootstrap process is finished, the Gateway may, at any time, send a part or the complete description of each device to be controlled by the CMS. The Gateway is responsible for assigning a unique TALQ address to each of its devices and of sending all configuration attributes defined for each of them. The CMS shall respond with HTTP CREATED once any such message is successfully received.



JSON Response

```
Request URL: https://<cmsUri>/devices?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-version": "2.1.0"
}
Request Content: [
 {
    "address": "110a8321-e34c-112p5-b567-566655748356",
    "name": "<deviceName>",
    "class": "<deviceClassName>",
    "functions": [
      {
         "id": "basic001",
         "type": "BasicFunction",
         "displayName": {
           "value": "<displayName>"
         },
         "assetId": {
           "value": "<assetId>"
         },
         …
      },
      {
         "id": "lampActuator001",
         "type": "LampActuatorFunction",
         "lampTypeId": {
           "value": "<lampTypeId>"
         },
         "maintenanceFactor": {
           "value": 50
         },
         "maintenancePeriod": {
           "value": 12
         },
         …
      },
      …
    ]
  },
  …
]
```

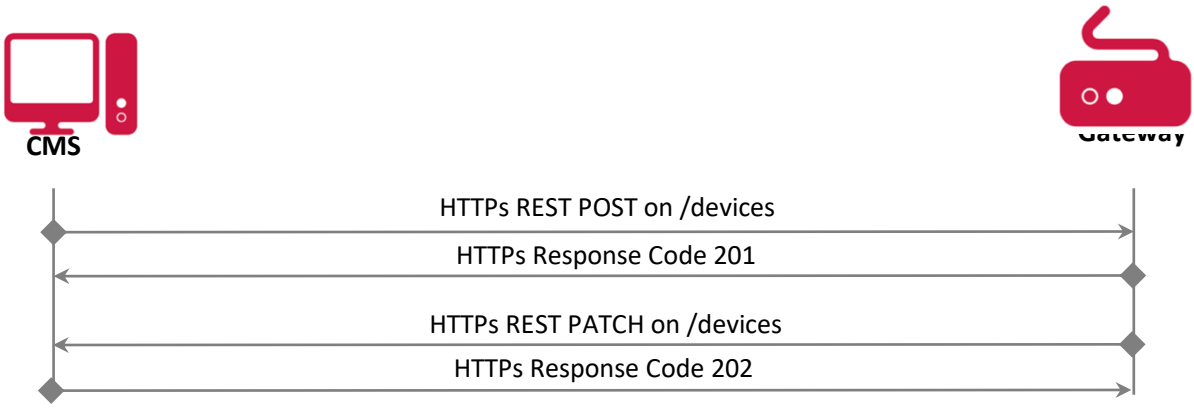
After receiving and processing a list of devices from the Gateway, the CMS may, at any time, send a PUT (or PATCH) request to change any of the configuration attributes of a device (except if such a configuration attribute is set as read-only), send a real time control command using a PUT (or PATCH) request on an

operational attribute (except if such an operational attribute is set as read-only) or send an on-demand data read using a GET request on a metering attribute.

5.3 Creating a device in a Gateway

The CMS may create a device in the Gateway by sending a POST request with the device class of the device (i.e. a device class that is supported by the Gateway) and the value of some or all of the configuration attributes of each function of the device. Such a POST may include a UUID or NIL UUID TALQ address.

If another device already exists in the Gateway with attributes which make it indistinguishable (by the Gateway) from the new device, the Gateway shall return the TALQ address of the existing device to the CMS if the CMS originally sent a NIL address. However, if the CMS originally sent a normal UUID address then the



Gateway shall respond with a CONFLICT response. The criteria for device uniqueness in a Gateway are specific to the ODN and is outside the scope of TALQ.

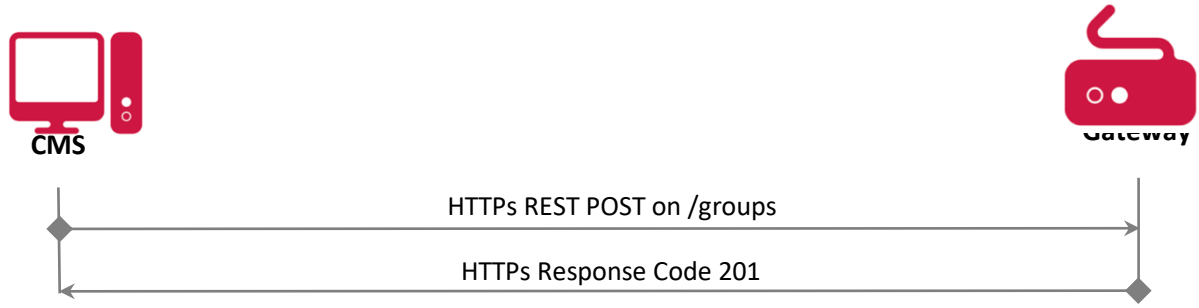
JSON Response

```
Request URL: https://<gatewayUri>/devices?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-version": "2.1.0"
}
Request Content: [
 {
    "address": "00000000-0000-0000-0000-000000000000",
    "name": "<deviceName>",
    "class": "<deviceClassName>",
    "functions": [
      {
         "id": "basic001",
         "type": "BasicFunction",
         "displayName": {
           "value": "<displayName>"
         },
         "assetId": {
           "value": "<assetId>"
         },
         …
      },
      {
         "id": "lampActuator001",
         "type": "LampActuatorFunction",
         "lampTypeId": {
           "value": "<lampTypeId>"
         },
         "maintenanceFactor": {
```

```
            "value": 50
          },
          "maintenancePeriod": {
            "value": 12
          },
          …
      },
      …
    ]
  },
  …
]
```

5.4 Creating a group

The group management service provides mechanisms to define and manage groups. Groups can be used across TALQ services whenever there is a need to address a set of devices and/or functions together. For instance, an override control command may be sent to a group of devices, or the same control program may be distributed to a group of actuators. A group may also include other groups as members. Groups can also be used to read multiple attribute values from multiple functions of multiple devices as well as in the configuration of the data loggers.
To create a group a POST request shall be sent by the CMS to the Gateway.



JSON Response

```
Request URL: https://<gatewayUri>/groups?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0",
}
Request Content: [
  {
    "address": "110a8321-e34c-112p5-b567-43126738927893",
    "members": [
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235810"},
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235811"},
       {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"}
    ],
    "purpose": "override"
  },
  …
]
```

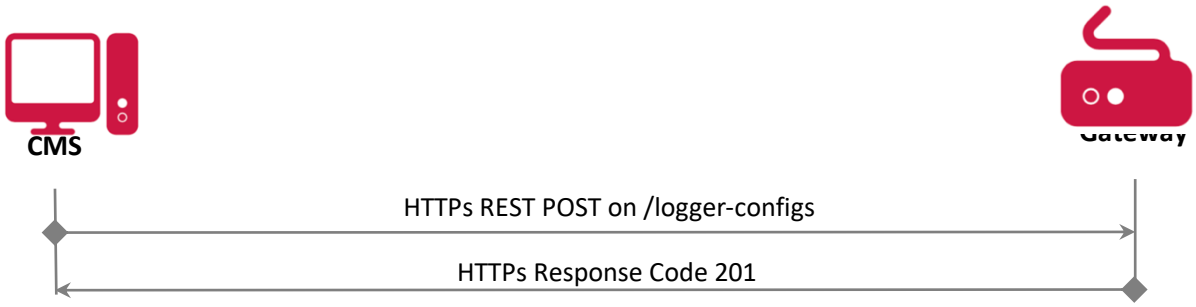
To update a group, the CMS shall send a PUT request with JSON payload replacing the current members with a new list. To delete the group in the Gateway the CMS shall send a DELETE request. To add one or more members to a group, the CMS shall send a REST PUT request to:
/groups/{address}/members
To delete one member from a group, the CMS shall send a REST DELETE request to:
/groups/{address}/members/{talqAddress}

5.5 Reporting log values

5.5.1 Configuring a data logger

The TALQ Specification does not specify how physical devices shall carry out measurements on a specific ODN, nor how they should communicate the sampled measurements and status attributes to the Gateway. However, the Gateway shall ensure the relevant attributes are logged according to the recording mode set by the CMS for every data logger. TALQ provides three recording modes:

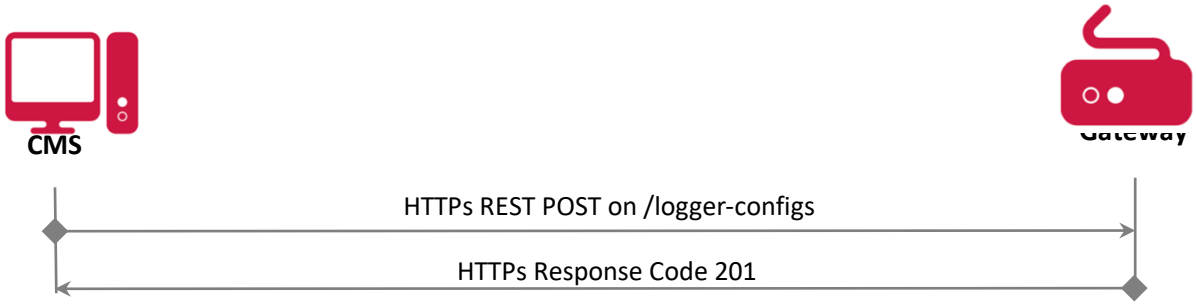
- **Vendor recording mode**: in this mode, attributes shall be logged using a sampling strategy defined by the Gateway vendor. This recording mode is the default one and doesn't require any specific configuration.



JSON Response

```
Request URL: https://<cmsUri>/logger-configs/<loggerId>?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
    "address": "110a8321-e34c-112p5-b567-1234567890",
    "sourceAddresses": {
      "whiteList": [
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
       {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"}
        …
      ]
    },
    "recordingMode": {
       "type": "VendorRecordingMode",
       "content": [
         {
           "functionType": "ElectricalMeterFunction",
           "attributes": [
             {
               "name": "totalActiveEnergy"
             },
             …
           ]
         },
         …
       ]
    },
    "reportingMode": { //ScheduledReportingMode
      "times": ["09:00:00"],
      "randomTime": 60
    }
}
```

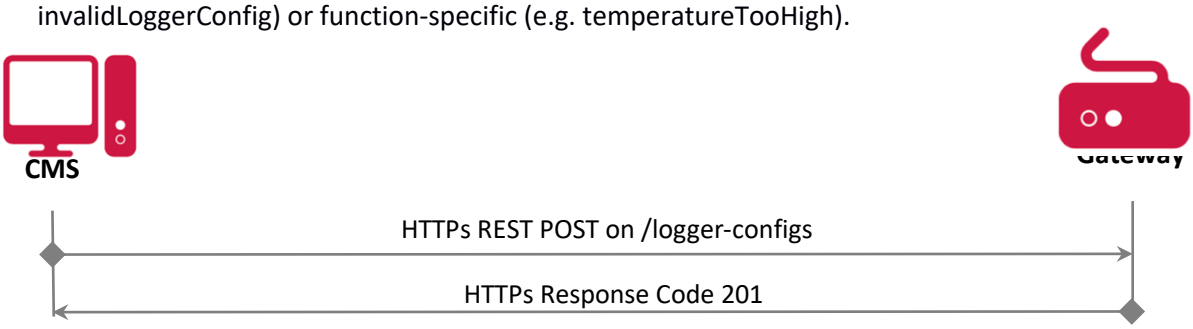
- **Periodic recording mode**: in this mode, attributes shall be logged using a time-based profile. A single logger shall have one or more sampling profiles for a single set of attributes. Each sampling profile defines one or more instants in time when a sample of all these attributes shall be logged.



JSON Response

```
Request URL: https://<cmsUri>/logger-configs/<loggerId>?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
    "address": "110a8321-e34c-112p5-b567-1234567890",
    "sourceAddresses": {
      "whiteList": [
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
       {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"},
…
      ]
    },
    "recordingMode": {
       "type": "PeriodicRecordingMode",
       "content": [
         {
           "functionType": "ElectricalMeterFunction",
           "attributes": [
             {
               "name": "totalActiveEnergy"
             },
             …
           ]
         },
         …
       ],
       "samplingProfile": {
         "samplingStartTime": "2013-04-01T00:00:00Z",
         "samplingPeriod": "P1D"
       }
    },
    "reportingMode": { //ScheduledReportingMode
      "times": ["09:00:00"],
      "randomTime": 60
    }
}
```

- **Event recording mode**: In this mode, the data logger shall record events which occur within the ODN. The TALQ defined events may be generic (e.g. invalidAddress), service-specific (e.g. invalidLoggerConfig) or function-specific (e.g. temperatureTooHigh).



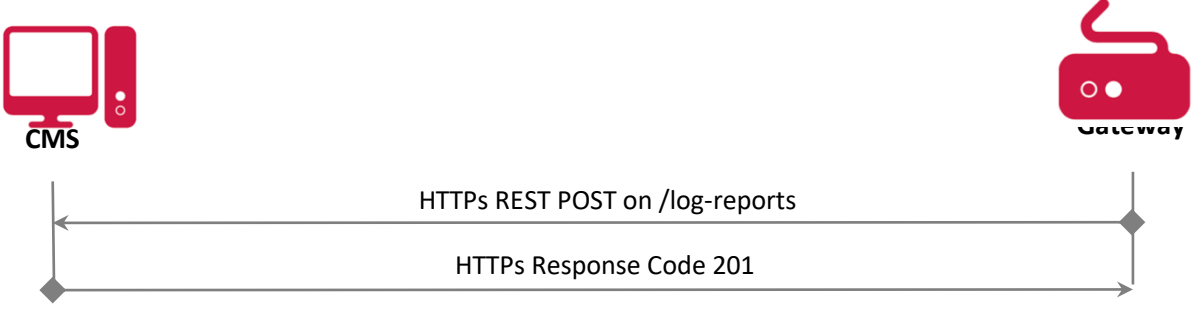
JSON Response

```
Request URL: https://<cmsUri>/logger-configs/logger001?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
    "address": "logger001",
    "sourceAddresses": {
      "whiteList": [
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
       {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"},
        …
      ]
    },
    "recordingMode": {
      "type": "EventRecordingMode",
      "sourceEvents": {
        "whiteList": [
          "invalidCalendar",
          …
        ]
      }
    },
    "reportingMode": { //ScheduledReportingMode
      "times": ["09:00:00"],
      "randomTime": 60
    }
}
```

5.5.2 Reporting log values

The reported data log shall include only entries that have not yet been successfully reported. If a Gateway cannot send all the data in a single REST request (and associated JSON payload), it shall send multiple messages. If the REST request is not acknowledged by the CMS, the Gateway shall retransmit it together with any new log records at the next scheduled reporting time. A Gateway may retry sending the REST request (and associated JSON payload) according to the "retryTimes" attribute set in the logger configuration.

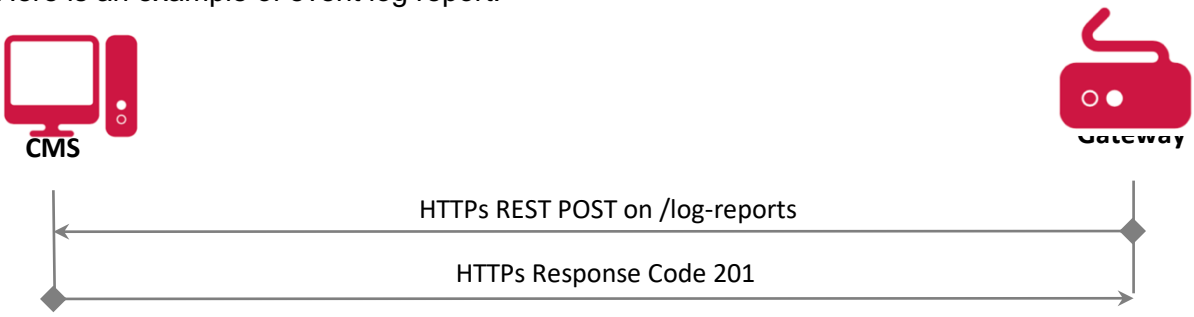
Below is an example of a periodic log report:



JSON Response

```
Request URL: https://<cmsUri>/log-reports?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "address": "logger001",
  "entries": [
    {
       "type": "AttributeLogData",
       "srcAddress": "110a8321-e34c-112p5-b567-566655685976",
       "timestamp": "2017-10-16T07:40:00Z",
       "attributeName": "totalActiveEnergy",
       "attributeValue": {
           "value": 130934.5
           "timestamp": "2017-10-16T07:33:16Z",
       }
    },
    {
       "type": "AttributeLogData",
       "srcAddress": "110a8321-e34c-112p5-b567-566655685976",
       "timestamp": "2017-10-16T07:40:00Z",
       "attributeName": "totalActivePower",
       "attributeValue": {
           "value": 153.3
           "timestamp": "2017-10-16T07:33:55Z",
       }
    }
   ]
}
```

Here is an example of event log report:



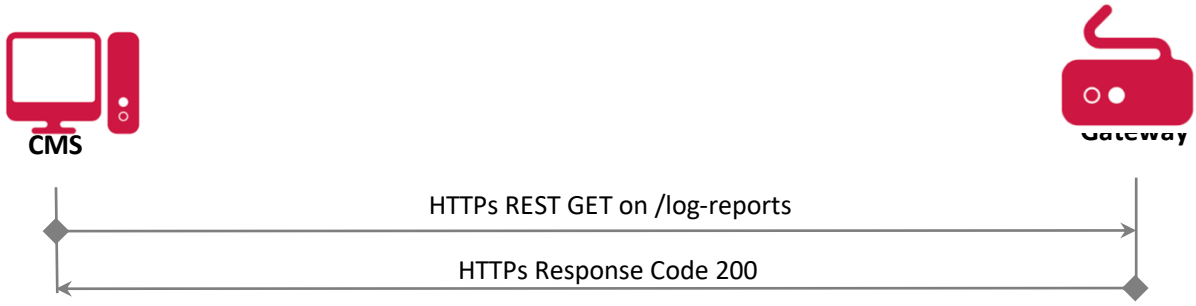
JSON Response

```
Request URL: https://<cmsUri>/log-reports?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "address": "logger002",
  "entries": [
    {
       "cmsRefId": "<cmsReference>",
       "data": {
         "type": "EventLogData",
         "eventType": "changingRelease",
         "srcAddress": "110a8321-e34c-112p5-b567-566655648453"
         "timestamp": "2018-02-16T07:33:16Z",
       }
    },
```

```
    {
       "cmsRefId": "<cmsReference>",
       "timestamp": "2018-02-16T08:26:14Z",
       "data": {
         "type": "EventLogData",
         "eventType": "releaseChanged",
         "srcAddress": "110a8321-e34c-112p5-b567-566655648453"
         "timestamp": "2018-02-16T07:33:16Z",
       }
    },
    …
  ]
}
```

5.5.3 Asking for a log report

At any time, the CMS may ask the Gateway for a log report. The Gateway shall answer by sending the current content of the log report. Here is an example of an "ad-hoc" log report query:



JSON Response

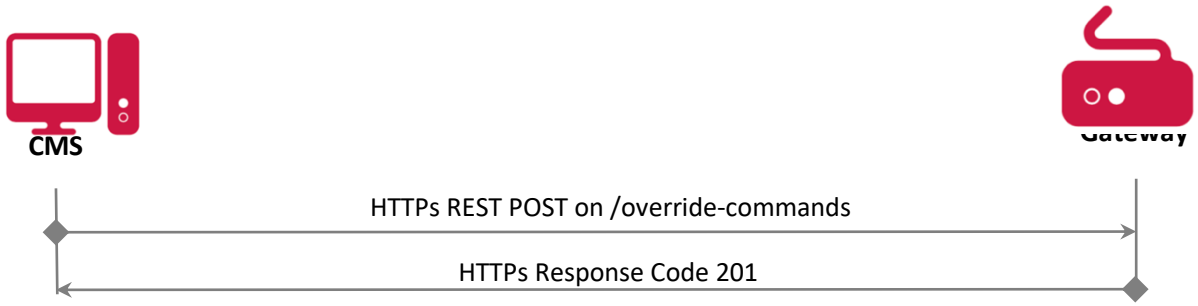
```
Request URL: https://<gatewayUri>/log-reports/logger001?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: GET
Status Code: "200 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Response Content: {
  "address": "logger001",
  "entries": [
    {
       "type": "AttributeLogData",
       "srcAddress": "110a8321-e34c-112p5-b567-566655685976",
       "timestamp": "2017-10-16T07:40:00Z",
       "attributeName": "totalActiveEnergy",
       "attributeValue": {
           "value": 130934.5
           "timestamp": "2017-10-16T07:33:16Z",
       }
    },
  ]
}
```

5.6 Sending an override command

A command defines a type of control action that can be applied to a TALQ function. Commands may be generated by a manual override action or by a control program. For example, to send a light command by an override action, such as a 50% dimming command to a lamp actuator function on a device, a CMS shall send a POST request to *gatewayUri/override-commands* associated with a JSON payload including the addresses of target devices or groups and the command with its parameters such as state (e.g. dimming level = 50%) and expiration date.

Sending override commands is the preferred way to act on a device (e.g. switching or dimming a lamp actuator) rather than updating the value of the associated operational attribute of the device, providing greater flexibility.

Here is an example of an override command from a CMS:



JSON Response

```
Request URL: https://<gatewayUri>/override-commands?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "addresses": [
     {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
     {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"}
    …
  ],
  "targetCommand": {
    "reason": "override",
    "cmsRefId": "<cmsReference>",
    "expiration": "2017-10-16T03:33:12Z",
    "rampToLevelTime": 0.5,
    "rampFromLevelTime": 2.0,
    "state": {
     "name": "levelState",
     "value": 50
  }
}
```

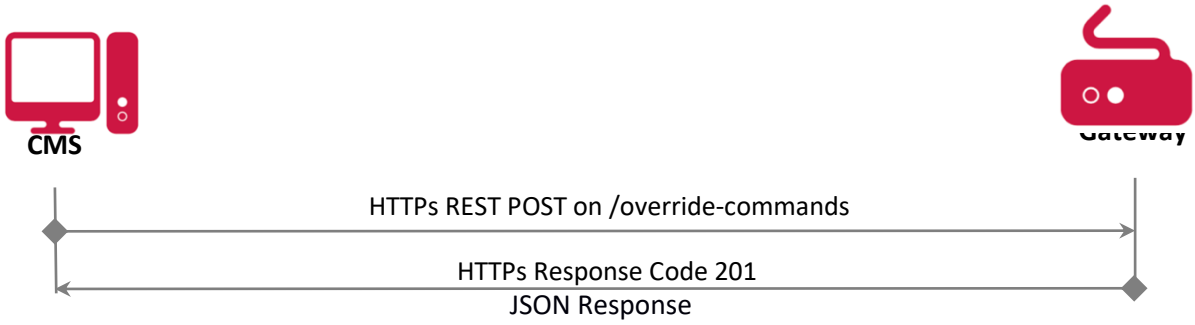
The "reason" parameter of a command indicates that the command was triggered by a manual override, a sensor or a control program. Permitted values are: unknown, default, override, sensor or program.

The "expiration" parameter is used to set a time to stop an override action. After the expiration of an override command, the system should go back to the state defined by the active control program. If not specified, there is no expiration for the override command.

The "rampToLevelTime" is the time (in seconds) taken for the value to ramp to the specified level after the reception of this override command request.

The "rampFromLevelTime" is the time (in seconds) taken for the value to ramp to the specified level. The change will be finished rampFromLevelTime seconds after the, in this example, expiration date time.

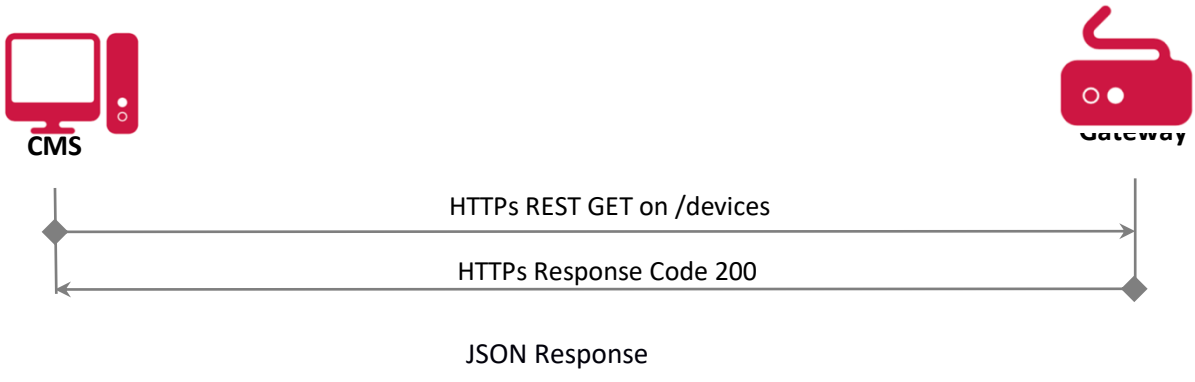
To resume an override command, the CMS shall send a POST request to the Gateway to resume one or more devices to their previous operation mode (e.g. control program). Here is an example of an override command resume:




```
Request URL: https://<gatewayUri>/override-commands?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "addresses": [
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
       {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"},
    …
  ]
}
```

5.7 Reading attribute values

At any time after the bootstrap process, the CMS may read an attribute value on the Gateway by sending a GET request on the device/function/attribute resource. For instance, a GET request on the "actualLightLevel" attribute of the lamp actuator function of a device should return the actual value of the light level on that device together with the timestamp at which that value was measured. Although the attribute's timestamp is not a required property as per the data model OAS file, it is considered a key protocol data for interoperability, and so a mandatory requirement to certify a product.




```
Request URL:
https://<gatewayUri>/devices/76472384789235810/lampActuator001/actualLightLevel?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: GET
Status Code: "200 OK"
Request Header: {
```

```
  "talq-api-version": "2.4.0"
}
Request Content: {
  "value": 23,
  "timestamp": "2017-10-16T07:33:16Z"
}
```

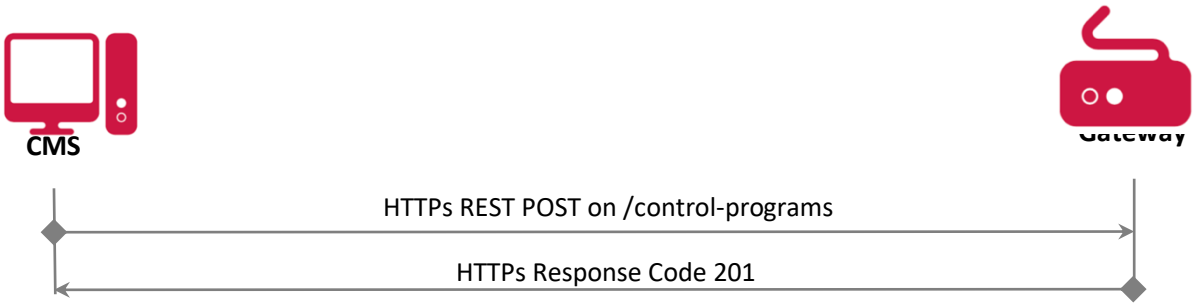
5.8 Assigning a control program to a group of devices

5.8.1 Creating a control program

A control program is a generic mechanism, or template, to automate the control of one or more devices within a 24hr period. It gathers a set of commands and is associated with calendars to indicate the target devices and when (i.e. which days) these commands should be executed.
A Control Program is a combination of control actions (i.e. commands) and trigger conditions that define when these commands should be executed on the target devices. The control program consists of the following main components:

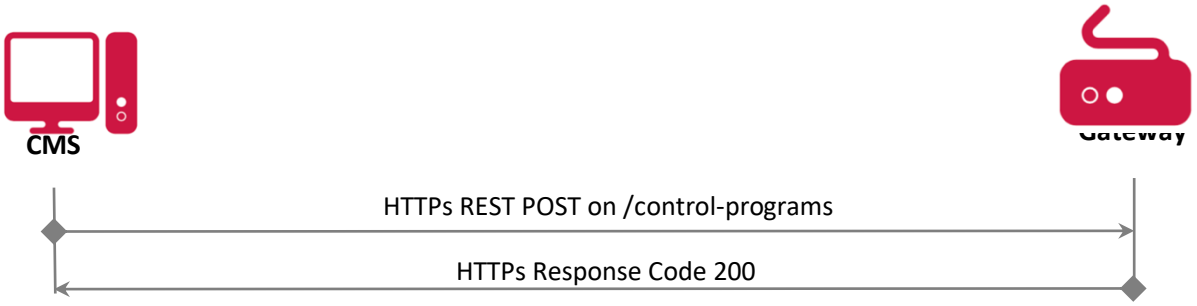
- **Active period**: a condition that defines the periods during a day in which the control program is active. In other words, it is a condition that determines when the target devices shall be under the control of the corresponding program. If no active period is defined within a control program, the program is active during the whole day. Different active periods are possible, such as fixed time periods (start and end times), sunrise-sunset times, photocell triggered and sensor triggered. By default, an actuator (e.g. generic actuator or lamp actuator) shall be OFF outside the active periods. Types and parameters of active periods are defined in the TALQ data model file.
- **Fixed time control**: it defines control commands for specific times during the day. It includes a start time and the command to be applied. A control program may contain one or more fixed time control elements. The end time of a fixed time control element is determined by the start time of the next fixed time control element in the control program. The next fixed time control element is with respect to the time order of the elements. The fixed time control entries within a program shall be included in ascending order to allow the next element to be easily identified. No two entries shall have exactly the same start time. Each parameter of fixed time control is defined in the TALQ data model file.
- **Dynamic control**: it defines control commands based on external trigger conditions (e.g. sensors) as well as a period (start and end times), which determines when the dynamic control is active during the day. The control commands may be specified in the program and triggered by an external source (e.g. sensor) or both control command and trigger may be specified by an external source. Furthermore, the dynamic control element also defines an operation effect (set, min, max, add, subtract, multiply) that the control action shall have on the current command. For example, if presence is detected by a sensor, light level could be configured by a control program to increase by 25% (add effect) or be set to 75% (set effect) or be set to 75% except if current level is already above that value (max effect).
  Multiple dynamic control elements may be active simultaneously and priority is determined by the order in which they are defined in the control program structure. Each parameter of dynamic control is defined in the TALQ data model file.
  Dynamic control enables use cases such as:
  - Dynamic outdoor lighting: automatically adjusting light levels on groups of streetlights when vehicles and/or pedestrians are detected or depending on vehicle count or speed.
  - Water irrigation: irrigating plants only when humidity is low and depending on sun and temperature.
  - Dynamic city information panel: displaying automatic messages on city information panels depending on traffic and/or pollution measurements.

To create a control program, the CMS shall send a REST POST request to the Gateway, examples are shown below:



JSON Response

```
Request URL: https://<gatewayUri>/control-programs?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "id": "<controlPrgramAddress>",
  "activePeriods": [
    {
      "type": "ActivePeriodAbsolute",
      "startTime": "21:30:00",
      "endTime": "07:00:00"
    },
    …
  ],
  "fixedTimeControls": [
      {
        "startTime": "00:00:00",
        "command": {
          "state": {
            "name": "LevelState",
            "value": 70
          }
        }
      },
      {
        "startTime": "05:00:00",
        "command": {
          "state": {
            "name": "LevelState",
            "value": 100
          }
        }
      },
      {
        "startTime": "23:00:00",
        "command": {
          "state": {
            "name": "LevelState",
            "value": 70
          }
        }
      }
  ]
}
```



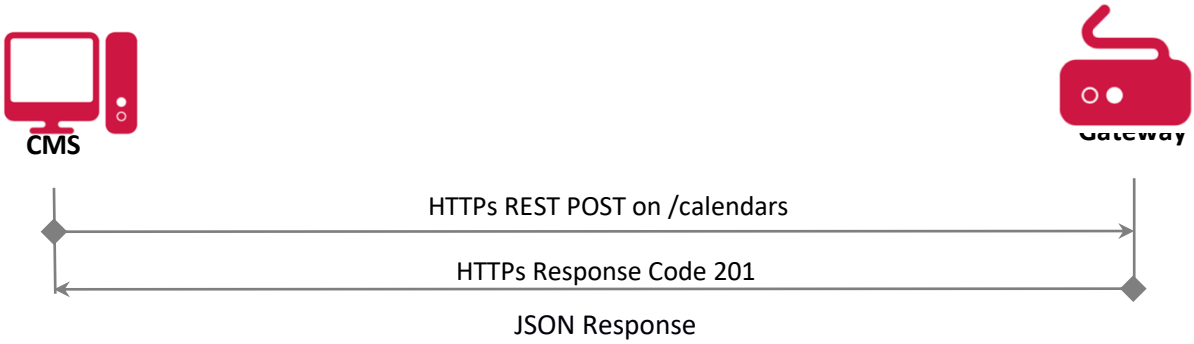
JSON Response

```
Request URL: https://<gatewayUri>/control-programs?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "id": "<controlPrgramAddress>",
  "activePeriods": [
    {
      "type":"ActivePeriodLightSensor",
      "sensor": ["<lightSensorTALQAddress>", …],
      "onIlluminance": 0.2,
      "offIlluminance": 0.6
    },
    …
  ],
  "fixedTimeControls": [
      {
        "startTime": "00:00:00",
        "command": {
          "state": {
            "name": "LevelState",
            "value": 70
          }
        }
      },
      {
        "startTime": "05:00:00",
        "command": {
          "state": {
            "name": "LevelState",
            "value": 100
          }
        }
      },
      {
        "startTime": "23:00:00",
        "command": {
          "state": {
            "name": "LevelState",
            "value": 70
          }
        }
      }
  ],
  "dynamicControl": [
      {
        "period": {
          "startTime": "05:30:00",
          "endTime": "16:00:00"
        }
        "effect": {
          "type": "ExternalControlEffect",
          "operation": "max",
          "holdTime": 10.0,
          "source": {
            "sensor": ["myTrafficSensorTALQAddress", …]
```

```
            }
          }
        }
      },
      …
  ]
}
```

5.8.2 Creating a calendar

A calendar is a set of rules that describe which days and which control program shall be applied to which device or groups of devices. The combination of calendars and control programs provides the flexibility necessary to implement schedule-based control which may be different for week-days and week-ends, in different seasons, during vacation time or on special days. Day-based and date-based rules can be programmed in the calendar and a control program shall be assigned to each rule. Rules priorities (e.g. between the first Thursday of the month and the 3rd of June if the 3rd of June is the first Thursday of the month) is defined by the order in the JSON payload structure.
The create a calendar, the CMS shall send a POST request to the Gateway as the example below:




```
Request URL: https://<gatewayUri>/calendars?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: [
  {
    "id": "myCalendar4",
    "rules": [
      {
        "startDate": "2018-07-18",
        "endDate": "2018-07-25",
        "program": "myControlProgram3"
      },
      {
        "condition": {
           "type": "ccDate",
           "dates": [
             {
                "start": "--10-16"
             }
           ]
        },
        "program": "myControlProgram2"
      },
    ],
    "defaultProgram": "myControlProgram1"
  }
]
```

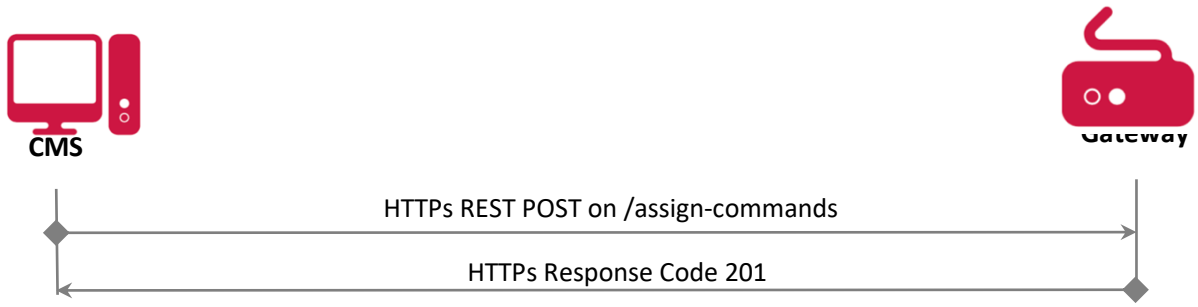
5.8.3 Assigning a calendar to devices

To assign a calendar to TALQ functions in a device or a group of devices, a CMS shall send a POST request to *gatewayUri/assign-commands* associated with a JSON payload that shall include the TALQ addresses of the

devices or group of devices, and the address of the entity to be assigned. If no specific function is indicated in the address, the command shall apply to a group of devices indicated in the JSON payload. Any calendar associated with the address(es) can be de-associated by setting the "*entity*" JSON element as an empty element.

Sending assign commands is the preferred way to act on a device (e.g. assigning a calendar to a lamp actuator) rather than updating the value of the associated operational attribute of the device, providing greater flexibility.

Asynchronous and synchronous behaviors are allowed. If a synchronous response is received (201-Created) the assignment is considered finished and thus the related devices updated. When an asynchronous response is returned (202-Accepted), the assignment is considered being processed and the CMS expects a PATCH request, or several, from the Gateway sometime in the future. The PATCH request(s) from the Gateway will include the list of all the affected devices.



JSON Payload

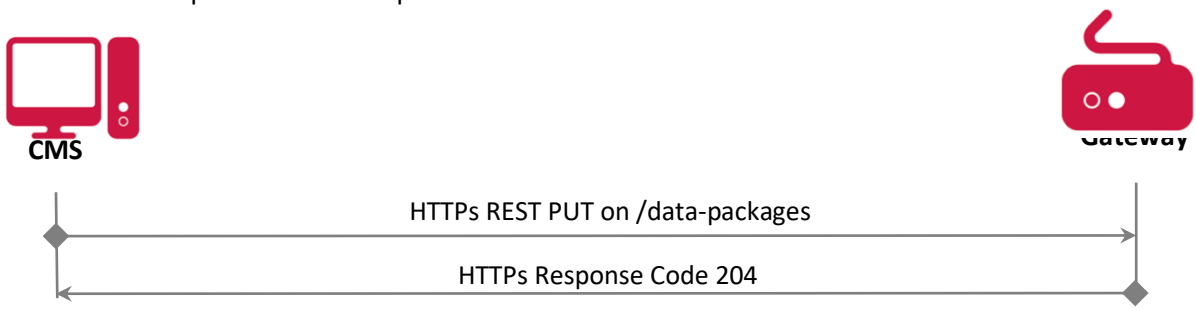
```
Request URL: https://<gatewayUri>/assign-commands?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: POST
Status Code: "201 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "addresses": [
       {"resource": "devices", "address": "110a8321-e34c-112p5-b567-76472384789235809"},
       {"resource": "groups", "address": "110a8321-e34c-112p5-b567-36423054023940392"}
     …
  ],
  "entity": {"resource": "calendars", "address": "calendar1"}
}
```

5.9 Sending a firmware update

To indicate a firmware update to devices or to the Gateway, the CMS shall send a PUT request on the /data-packages resource of the Gateway with some JSON payload that contains:

- A package identifier for the firmware package,
- A release identifier for the firmware package,
- The URI of firmware package to download the new binary content.

Here is an example of firmware update:



JSON Response

```
Request URL: https://<gatewayUri>/data-packages?clientAddress=110a8321-e34c-112p5-b567-566655648453
Request Method: PUT
Status Code: "204 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Request Content: {
  "packageId": "<packageId>",
  "releaseId": "3.2.4",
  "downloadUri": "…"
}
```

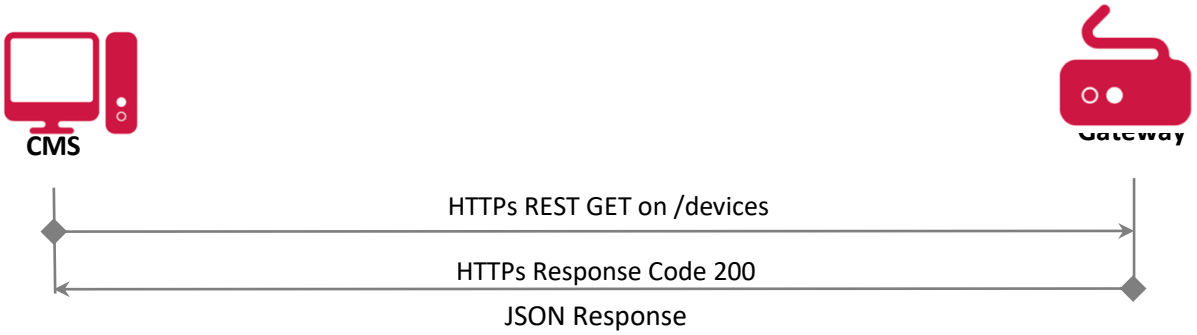
It is the responsibility of the Gateway to accept or refuse the firmware update and, if accepted, to execute it (e.g download and install in the appropriate device). It is not in the scope of TALQ to define the execution process. The Gateway may then send one or more of the following events, using the data collection service, to indicate progress:

- releaseMismatch when the release indicated in the JSON payload is not compatible with some hardware on the ODN or not recognized as a valid release by the Gateway.
- changingRelease when the Gateway has started the process of changing the release on the target devices.
- changeReleaseFailure when the release change failed, to inform the CMS that the release identifier on the target devices remained the release identifier prior the operation.
- releaseChanged when the release is downloaded and running on each target device.

5.10 Reading a partial list of devices

The optional devices pagination feature can be used for reading a partial list of devices from a large set of them.
Pagination is based on two query parameters at the request: offset and limit. The server will answer with a partial list of devices with a 'limit' number of devices starting at 'offset' index. If none of the parameters are present, pagination won't be used and the result will include the whole list of devices.
Pagination requires an implied ordering/indexing, although it is outside TALQ scope and will be the Gateway responsibility.
When pagination is active, the response headers also include useful information related to it.

The Gateway shall announce if it supports pagination by setting devicePaginationSupported=true in the Configuration Service announcement.
Below there is an example where the CMS asks the GW for the first 10 devices. The server answers with a list of devices and metadata information at the response headers related to pagination.




```
Request URL: https://<gatewayUri>/devices?clientAddress=110a8321-e34c-112p5-b567-566655648453&offset=0&limit=10
Request Method: GET
Status Code: "200 OK"
Request Header: {
  "talq-api-version": "2.4.0"
}
Response Header: {
  " pagination-offset": 0,
  " pagination-defaultLimit": 20,
  " pagination-count": 10,
  " pagination-totalCount": 100,

}
Response Content: [
   {
      "address": "D5144581-E2B4-4116-AC29-55357714768D", // Index: 0
      "deviceClass": "lampClass",
      "functions":[
      …
      ]
   },
   …
   {
      "address": "9994E758-38C6-4E7B-83A0-731B5AED7C72", // Index: 9
      "deviceClass": "lampClass",
      "functions":[
      …
      ]
   },
]
```

6 REFERENCES

6.1 Normative references

- [RFC 2616] Hypertext Transfer Protocol -- HTTP/1.1 (http://www.tools.ietf.org/html/rfc2616)
- [RFC 6202] Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP
- [RFC 2818] HTTP Over TLS (http://tools.ietf.org/html/rfc2818)
- [RFC 5246] The Transport Layer Security (TLS) Protocol Version 1.2 (http://tools.ietf.org/html/rfc5246)
- [RFC 2396] Uniform Resource Identifiers (URI): Generic Syntax (http://www.ietf.org/rfc/rfc2396)
- [RFC 4122] A Universally Unique IDentifier (UUID) Namespace (https://tools.ietf.org/html/rfc4122)
- [Krawczyk97] H. Krawczyk, M. Bellare and R. Canetti, HMAC: Keyed-Hashing for Message Authentication, RFC 2104, February 1997.
- [Chown02] P. Chown, AES Ciphersuites for TLS, RFC 3268, June 2002.
- [Cooper08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280, May 2008.
- [ISO8601] ISO 8601:2004 ISO (International Organization for Standardization), Data elements and interchange formats – Information interchange -- Representations of dates and times.
- [RSA78] Rivest, R.; A. Shamir; L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21 (2): 120–126, 1978.
- [RFC 1630] Universal Resource Identifiers in WWW (http://tools.ietf.org/html/rfc1630)
- [RFC 1738] Uniform Resource Locators (URL) (http://tools.ietf.org/html/rfc1738)
- [RFC 1808] Relative Uniform Resources Locators (http://tools.ietf.org/html/rfc1808)
- [RFC 2717] Registration Procedures for URL Scheme Names (http://tools.ietf.org/html/rfc2717)
- [IANA] IANA time zone database (Olson database). http://www.iana.org/time-zones.
- [POSIX] Open Group for posix systems. http://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap08.html
- [JSONSCHEMA] More details about specifications: http://json-schema.org/specification.html

6.2 Informative references

- [Fielding] Fielding, Roy Thomas (2000), "Architectural Styles and the Design of Network-based 668 Software Architectures", Doctoral Dissertation, University of California, Irvine.
- [LSD64-2012] Lighting Control Terminology, A NEMA Lighting Systems Division Document (LSD 64-2012).

TALQ Consortium
445 Hoes Lane
Piscataway
NJ 08854, USA
info@talq-consortium.org
www.talq-consortium.org